Concise Linear Regression Implementation

20 July 2021 13:53

1) Generating the dataset

```
import numpy as np
import tensorflow as tf

def synthetic_data(w, b, num_examples):
    X = tf.zeros((num_examples, w.shape[0]))
    X+= tf.random.normal(shape = X.shape)
    y = tf.matmul(X, tf.reshape(w, (-1,1))) + b
    y+=tf.random.normal(shape=y.shape, stddev = 0.01)
    y = tf.reshape(y, (-1,1))
    return X, y

true_w = tf.constant([2, -3.4])
true_b = 4.2
features, labels = synthetic_data(true_w, true_b, 1000)
```

2) Reading the dataset

→ Instead of iterator, we can call upon an existing API in a framework to read data.

→ Arguments:
- features, labels
- batch_size
- is_train → data iterator to shuffle the data on each epoch

```
def load_array(data_arrays, batch_size, is_train = True):
    """Construct a Tensorflow data iterator"""
    dataset = tf.data.Dataset.from_tensor_slices(data_arrays)
    if is_train:
        dataset = dataset.shuffle(buffer_size = 1000)
    dataset = dataset.batch(batch_size)
    return dataset

batch_size = 10
data_iter = load_array((features, labels), batch_size)
```

We can use data_iter now as follows

We can use data_iter now as follows

python iterator

next(iter(data_iter))

first item from iterator

3) Defining the Model

- Although we should know how to implement a model from scratch, we need not reinvent the wheel every single time
- We can use the framework's predefined layers which allows us to focus on the layers used to construct the model rather than on their implementation

net → Instance of the Sequential class

→ Sequential class

↳ container for several layers that will be chained together

↳ Given input data

↳ Sequential passes it through the first layer in turn

using it as input for the second layer & so on

In keras, a fully connected layer is defined in the Dense class

For keras, we do not need to specify input shape for each layer → automatically inferred from input

```
net = tf.keras.Sequential()
net.add(tf.keras.layers.Dense(1))
```

↑ since only 1 scalar output is required

The initializers module in TensorFlow provides various methods for model parameter initialization

So while creating & adding layers to net:

```
initializer = tf.initializers.RandomNormal(stddev = 0.01)
net = tf.keras.Sequential()
net.add(tf.keras.layers.Dense(1, kernel_initializer = initializer))
```

↑ initialization is deferred

4) Defining the Loss Function :-

```
loss = tf.keras.losses.MeanSquaredError()
```

5) Defining the Optimisation Algorithm

```
trainer = tf.keras.optimizers.SGD(learning_rate=0.03)
```

6) Training

For some number of epochs, make a pass over the complete dataset (train_data) iteratively grabbing one minibatch of inputs and corresponding ground truth labels

For each minibatch:-

1) Generate the predictions & calculate the loss l

2) Calculate gradients by running backpropagation

3) Update model parameters by invoking our optimizer

```
num_epochs = 3
for epoch in range(num_epochs):
        for X,y in data_iter:
                with tf.GradientTape() as tape:
                        l = loss(net(X, training = True), y)
                grads = tape.gradient(l, net.trainable_variables)
                trainer.apply_gradients(zip(grads, net.trainable_variables))
        l = loss(net(features), labels)
        print(f'epoch {epoch+1}, loss {l:f}')

w = net.get_weights()[0]
print('error in estimating w', true_w - tf.reshape(w, true_w.shape))
b = net.get_weights()[1]
print('error in estimating b', true_b - b)
```

Classification Using Softmax Regression

20 July 2021 17:52

Regression → How much? How many?

Classification → Which one?

1) Classification :-

One Hot Encoding

eg cat → (1, 0, 0)

chicken → (0, 1, 0)

dog → (0, 0, 1)

$y \in \{(1,0,0), (0,1,0), (0,0,1)\}$

Hard assignment → Output category to which data belongs

Soft assignment → Output probability of belonging to each category

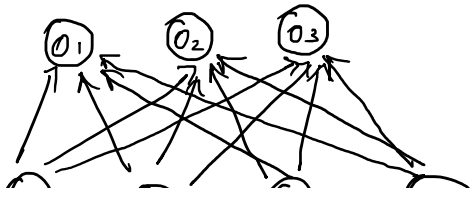
2) Network Architecture

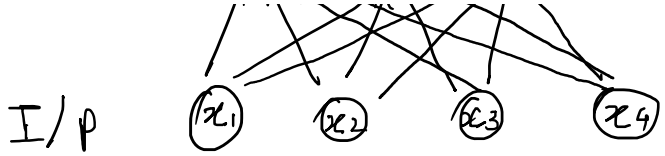
$$O_1 = x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1$$

$$O_2 = x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2$$

$$O_3 = x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3$$

O/P





In vector form

$$o = Wx + b \implies W \in \mathbb{R}^{3\times 4}$$

Parameterization Cost of Fully Connected Layers

For fully connected layers with d inputs & q outputs,

Parameterization Cost: $O(dq)$

Can be reduced to: $O(dq/n)$

where hyperparameter n can be flexibly specified by us to balance between parameter saving & model effectiveness in the real world

3) Softmax Operation

We need to interpret our outputs as probabilities

For this :-

1) Outputs must be non-negative

2) Outputs sum upto 1

3) Model remains differentiable

$$\hat{y} = \text{softmax}(o) \quad \text{where} \quad \hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}$$

$$\sum_j \hat{y}_j = 1 \quad \text{with} \quad 0 < \hat{y}_j \leq 1 \ \forall j$$

4) Vectorization for Minibatches

Number of inputs $\rightarrow d$

Number of examples $\rightarrow n$

Number of categories $\rightarrow q$

$$O = XW + b$$

$$\hat{Y} = \text{softmax}(O)$$

$$l(y, \hat{y}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j$$

$$l(y, \hat{y}) = -\sum_{j=1}^{q} y_j \log \frac{e^{o_j}}{\sum_{k=1}^{q} e^{o_k}}$$

$$= \sum_{j=1}^{q} y_j \log \sum_{k=1}^{q} e^{o_k} - \sum_{j=1}^{q} y_j o_j$$

$$= \log \sum_{k=1}^{q} e^{o_k} - \sum_{j=1}^{q} y_j o_j$$

$$\partial_{o_j} l(y, \hat{y}) = \frac{e^{o_j}}{\sum_{k=1}^{q} e^{o_k}} - y_j$$

$$= \text{softmax}(O)_j - y_j$$

The Image Classification Dataset

21 July 2021 10:20

MNIST Dataset

```
%matplotlib inline
import tensorflow as tf
from d2l import tensorflow as tf

d2l.use_svg_display()
```

1) Reading the dataset

↳ Load Fashion MNIST dataset into memory via the built-in functions

```
mnist_train, mnist_test = tf.keras.datasets.fashion_mnist.load_data()
```

features, labels

features, labels

Images from :-

1) 10 categories
2) 60,000 images in training dataset
3) 10,000 images in test dataset

```
len(mnist_train[0]), len(mnist_test[0])
```

```
mnist_train[0][0].shape
```

- Dimensions of the input image (28×28)
- Grayscale images — number of channels=1

For brevity, shape of any image with height h & width w → (h, w)

Images in category are associated with

categories → T-shirt, trousers, pullover, dress, coat, sandal, shirt, sneaker, bag & ankle boot.

```
def get_fashion_mnist_labels(labels): #@save
        """Return text labels for the Fashion MNIST dataset"""
        text_labels = [
        't-shirt', 'trouser', 'pullover', 'dress', 'coat', 'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
        return [text_labels[int(i)] for i in labels]
```

```
def show_images(imgs, num_rows, num_cols, titles=None, scale=1.5):  #@save
    """Plot a list of images."""
    figsize = (num_cols * scale, num_rows * scale)
    _, axes = d2l.plt.subplots(num_rows, num_cols, figsize=figsize)
    axes = axes.flatten()
    for i, (ax, img) in enumerate(zip(axes, imgs)):
        ax.imshow(img.numpy())
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        if titles:
            ax.set_title(titles[i])
    return axes
```

```
X = tf.constant(mnist_train[0][:18])
y = tf.constant(mnist_train[1][:18])
```

```
show_images(X, 2, 9, titles = get_fashion_mnist_labels(y));
```

2) Reading A Minibatch

```
batch_size = 256
train_iter =
tf.data.Dataset.from_tensor_slices(mnist_train).batch(batch_size).shuffle(len(mnist_train[0]))
```

```
timer = d2l.Timer()
for X,y in train_iter:
        continue
f'{timer.stop():.2f} sec'
```

3) Putting it all Together

```
def load_data_fashion_mnist(batch_size, resize = None):
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
        x_train = x_train.astype('float32')/255
        x_test = x_test.astype('float32')/255
        # lambda function to resize the image
        resize_fn = lambda X: (tf.image.resize_with_pad(X, resize, resize) if resize else X)
        x_train = x_train.map(resize_fn)
        x_test = x_test.map(resize_fn)
```

```
y_train = tf.cast(y_train, 'int32')
y_test = tf.cast(y_test, 'int32')

return ( tf.data.Dataset.from_tensor_slices((x_train,
y_train)).batch(batch_size).shuffle(len(x_train)), tf.data.Dataset.from_tensor_slices((x_test,
y_test)).batch(batch_size).shuffle(len(x_test)))
```

4) Loading the data, normalising & resizing data

```
batch_size = 256
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size)
```

5) Initializing Model Parameters

```
num_inputs = 784
num_outputs = 10

W = tf.Variable(tf.random.normal(shape = (num_inputs, num_outputs), mean = 0, stddev = 0.01))
b = tf.Variable(tf.zeros(num_outputs))
```

6) Defining the Softmax Operation

```
def softmax(X):
    X_exp = tf.exp(X)
    partition = tf.reduce_sum(X_exp, 1, keepdim = True)
    return X_exp/partition
```

7) Defining the Model

```
def net(X):
    return softmax(tf.matmul(tf.reshape(X, (-1, W.shape[0])), W) + b)
```

8) Defining the Loss Function

```
def cross_entropy(y_hat, y):
    return -tf.math.log(tf.boolean_mask(y_hat, tf.one_hot(y, depth = y_hat.shape[-1])))
```

9) Defining accuracy

```
def accuracy(y_hat, y):
    """ Compute the number of correct predictions """
    if len(y_hat.shape) > 1 and y_hat.shape[1] > 1:
        y_hat = tf.argmax(y_hat, axis = 1)
    cmp = tf.cast(y_hat, y.dtype) == y
    return float(tf.reduce_sum(tf.cast(cmp, y.dtype)))
```

accuracy(y_hat, y)/len(y)

```
def evaluate_accuracy(net, data_iter):
    """ Compute the accuracy for a model on a dataset. """
    metric = Accumulator(2)
    for X, y in data_iter:
        metric.add(accuracy(net(X), y), tf.size(y).numpy())
    return metric[0]/metric[1]

class Accumulator:
    """ For accumulating sums over `n` variables """
    def __init__(self, n):
        self.data = [0.0] * n

    def add(self, *args):
        self.data = [a + float(b) for a, b in zip(self.data, args)]
    def reset(self):
        self.data = [0.0] * len(self.data)

    def __getitem__(self, idx):
        return self.data[idx]

evaluate_accuracy(net, test_iter)

def train_epoch_ch3(net, train_iter, loss, updater):
    metric = Accumulator(3)
    for X, y in train_iter:
        y_hat = net(X)
        # Keras implementations for loss takes (labels, predictions)
        # instead of (predictions, labels) that users might implement
        # in this book, e.g. `cross_entropy` that we implemented above
        if isinstance(loss, tf.keras.losses.Loss):
            l = loss(y, y_hat)
        else:
            l = loss(y_hat, y)
        if isinstance(updater, tf.keras.optimers.Optimizer):
            params = net.trainable_variables
            grads = tape.gradient(l, params)
            updater.apply_gradients(zip(grads, params))
        else:
            updater(X.shape[0], tape.gradient(l, updater.params))
        l_sum = l * float(tf.size(y)) if isinstance(loss, tf.keras.losses.Loss) else
        tf.reduce_sum(l)
        metric.add(l_sum, accuracy(y_hat, y), tf.size(y))
        return metric[0]/metric[2], metric[1]/metric[2]
```

Utility Class for plotting data

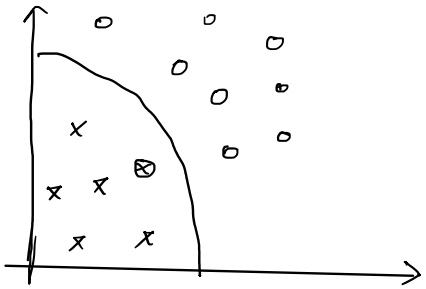
```
class Animator:  #@save
    """For plotting data in animation."""
    def __init__(self, xlabel=None, ylabel=None, legend=None, xlim=None,
                 ylim=None, xscale='linear', yscale='linear',
                 fmts=('-', 'm--', 'g-.', 'r:'), nrows=1, ncols=1,
                 figsize=(3.5, 2.5)):
        # Incrementally plot multiple lines
```

```
        if legend is None:
            legend = []
        d2l.use_svg_display()
        self.fig, self.axes = d2l.plt.subplots(nrows, ncols, figsize=figsize)
        if nrows * ncols == 1:
            self.axes = [self.axes,]
        # Use a lambda function to capture arguments
        self.config_axes = lambda: d2l.set_axes(self.axes[
            0], xlabel, ylabel, xlim, ylim, xscale, yscale, legend)
        self.X, self.Y, self.fmts = None, None, fmts
    def add(self, x, y):
        # Add multiple data points into the figure
        if not hasattr(y, "__len__"):
            y = [y]
        n = len(y)
        if not hasattr(x, "__len__"):
            x = [x] * n
        if not self.X:
            self.X = [[] for _ in range(n)]
        if not self.Y:
            self.Y = [[] for _ in range(n)]
        for i, (a, b) in enumerate(zip(x, y)):
            if a is not None and b is not None:
                self.X[i].append(a)
                self.Y[i].append(b)
        self.axes[0].cla()
        for x, y, fmt in zip(self.X, self.Y, self.fmts):
            self.axes[0].plot(x, y, fmt)
        self.config_axes()
        display.display(self.fig)
        display.clear_output(wait=True)
```

Naive Bayes Classification

12 September 2021 11:29

Decision Surfaces



→ Linear

Naive Bayes

Supervised Classification using Naive Bayes

- sklearn → scikit learn
  sklearn → naive bayes

Bayes Rule

$P(C) = 0.01$

Test : 90% it is positive if you have C
90% it is negative if you don't have C

Test = Positive ; Probability you have C

$P(T/C) = 0.9 \qquad P(C) = 0.01$

$P(T'/C') = 0.9 \qquad P(C') = 0.99$

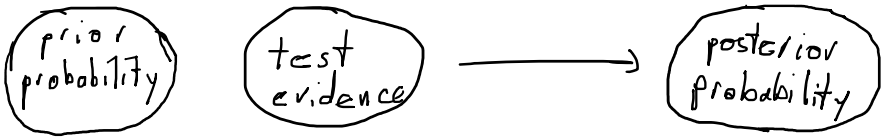
$$P(C/T) = \frac{P(C \cap T)}{P(T)}$$

$$= \frac{0.9 \times 0.01}{P(C)P(T/C) + P(C')P(T/C')}$$

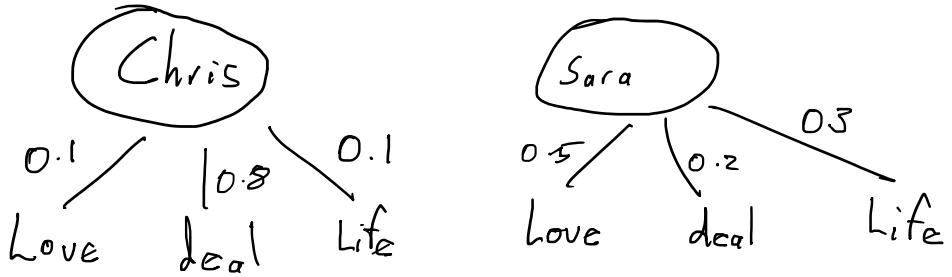
$$= \frac{0.9 \times 0.01}{0.01 \times 0.9 + 0.99 \times 0.1}$$

$$= 0.08333$$

Bayes Rule



Text Learning — Naive Bayes



Love Life !

$P(Chris) = 0.5$

$P(Sara) = 0.5$

$$P(C/L) = \frac{P(L/C)P(C)}{P(L/C)P(C) + P(L/\neg C)P(\neg C)}$$

$$= \frac{0.1 \times 0.5}{0.1 \times 0.5 + 0.5 \times 0.5}$$

$$= 0.1667$$

$$P(C/Life) = P(Life/C)P(C)$$

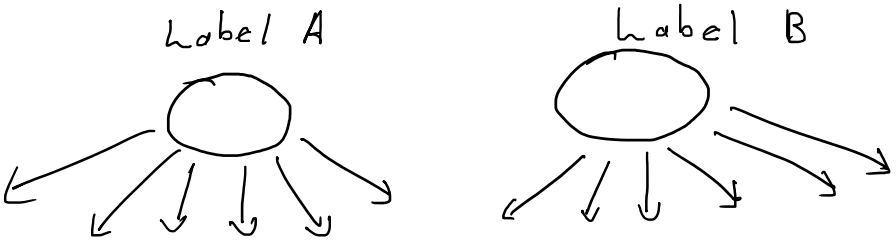
$$P(C/Life) = \frac{P(Life/C)\,P(C)}{P(Life/C)\,P(C) + P(Life/\neg C)\,P(\neg C)}$$

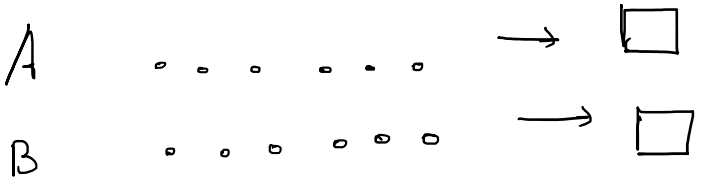
$$= \frac{0.1 \times 0.5}{0.1 \times 0.5 + 0.3 \times 0.5}$$

$$= 0.25$$

$$P(C/\text{"Life Deal"}) = \frac{0.1 \times 0.8 \times 0.5}{0.1 \times 0.8 \times 0.5 + 0.3 \times 0.2 \times 0.5}$$

$$= 0.47$$





Naive Bayes

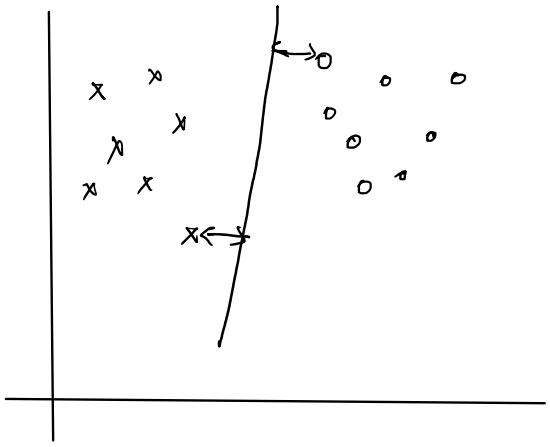
- Easy to implement
- Can break really in strange ways

  eg Chicago Bulls

  - word order is important

Support Vector Machines

12 September 2021 13:17

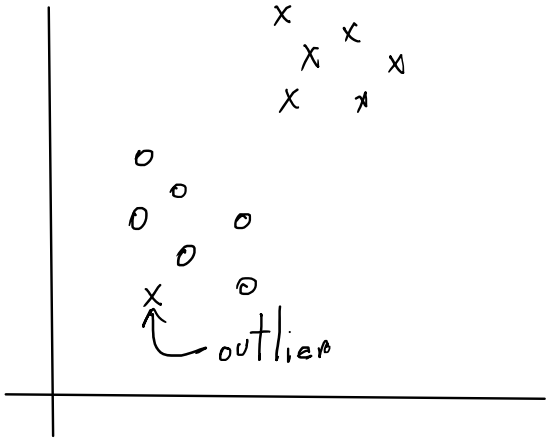


$\max_{\forall i} \left(\min\left(\min(p_i, l_i), \min(q_i, l_i) \right) \right)$

relative to both groups/classes

Maximizes distance to nearest point
= margin

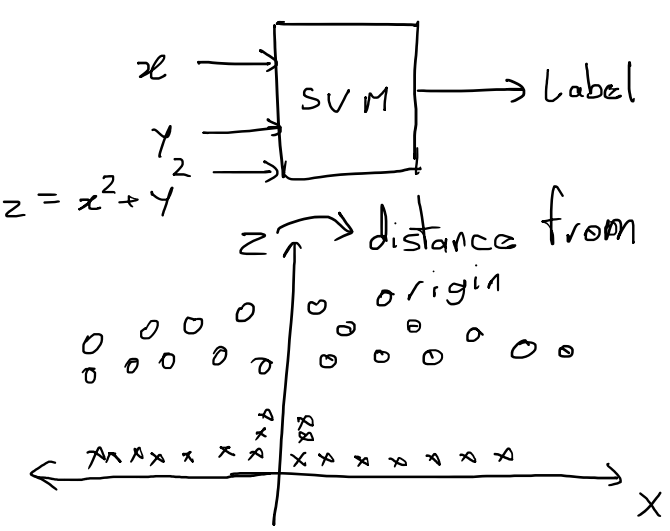
SVMs – Outliers

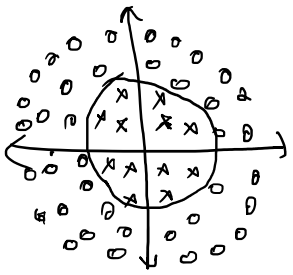


Tolerate outliers while maximizing distance

SVM can make really complex shapes
↳ Can give non linear decision boundaries

Features can be used to create non-linear decision boundaries





Kernel Trick

X, Y —Kernels→ X_1, X_2, X_3, X_4, X_5

Not separable — Separable

Solution ← Non linear separation

[sklearn.svm.SVC — scikit-learn 0.24.2 documentation]

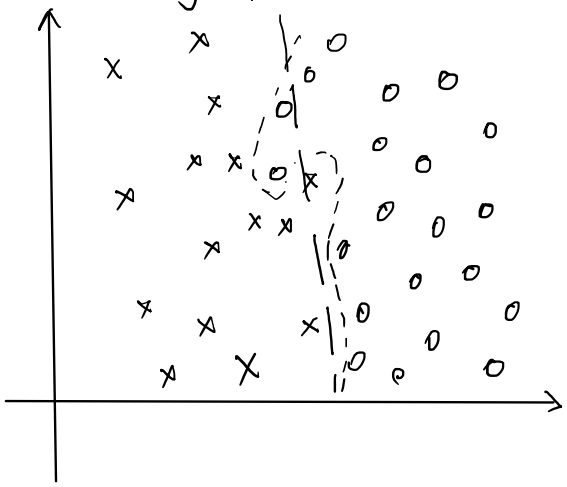
kernels available → linear, poly, rbf, sigmoid & more (precomputed, callable)

Parameters in machine learning

For an SVM

- → kernel — linear — rbf
- → C — 1000 — 1000
- → gamma

C → controls tradeoff b/w smooth decision boundary & classifying training points correctly



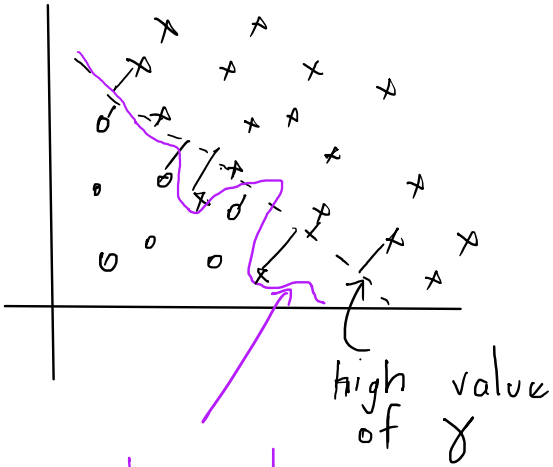
More curvaceous might not generalize well

C ↑ regularization ↓

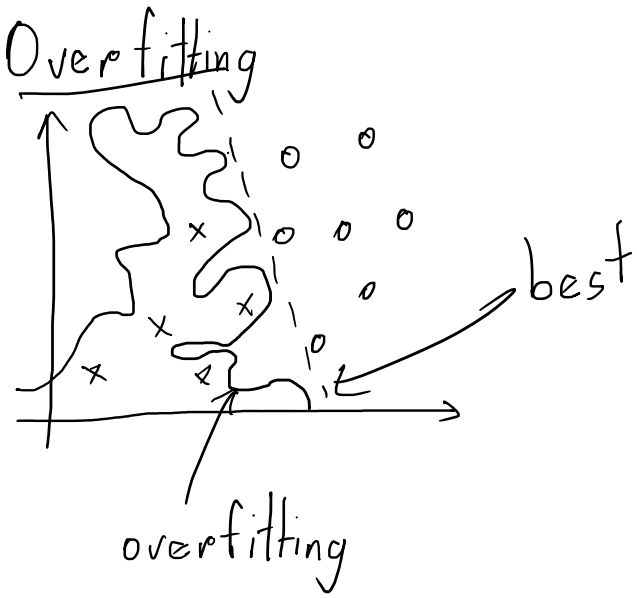
SVM γ parameter.

γ → defines how far the influence of a single training example reaches

low values → far
high values → close



Overfitting

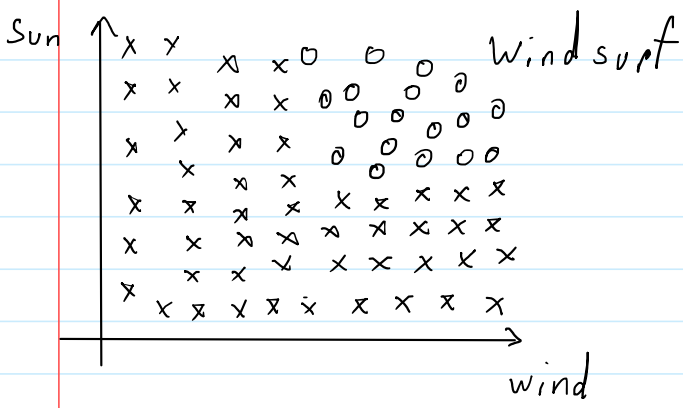


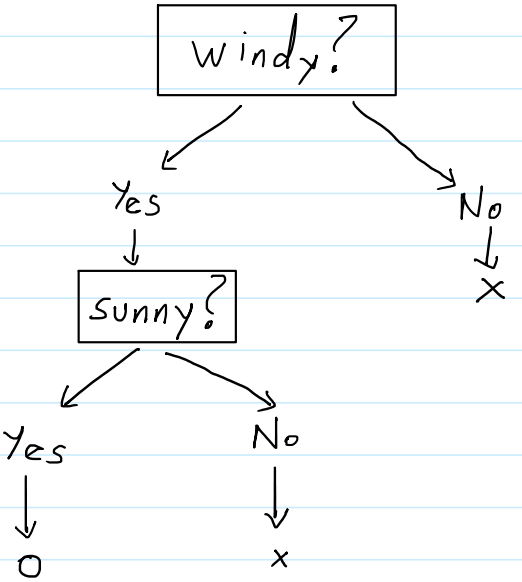
SVM

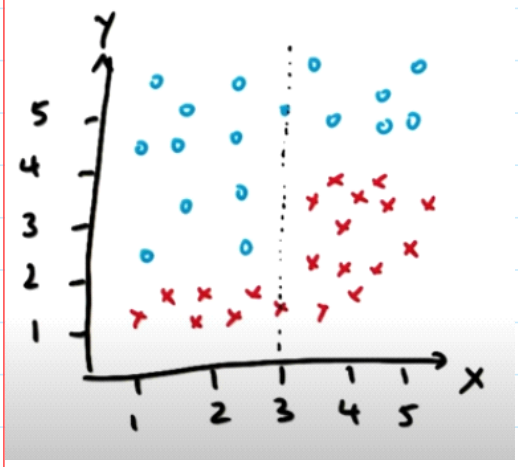
- ✓ → Complex domain → clear margin of separation
- X → Large datasets
- X → Overlapping → Naive bayes is better

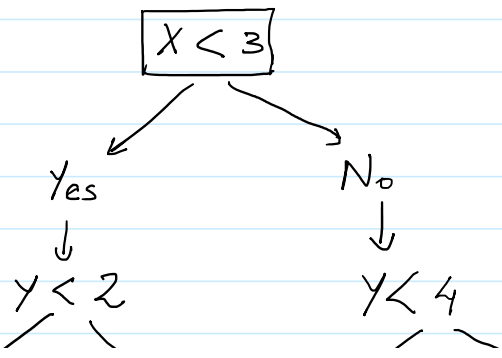
Decision Trees

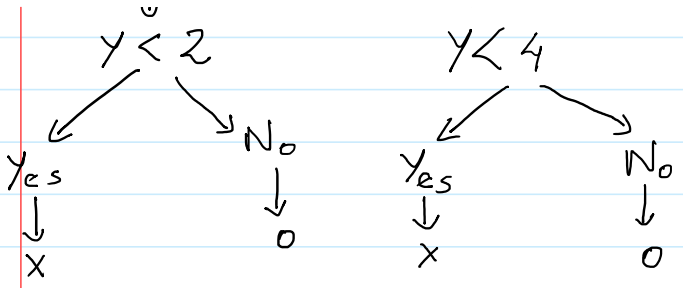
12 September 2021 21:43





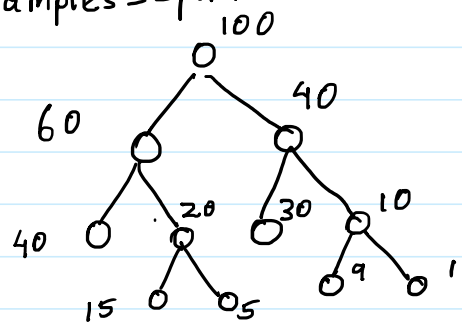






Parameter Tuning

min_samples_split ← default value ⇒ 2

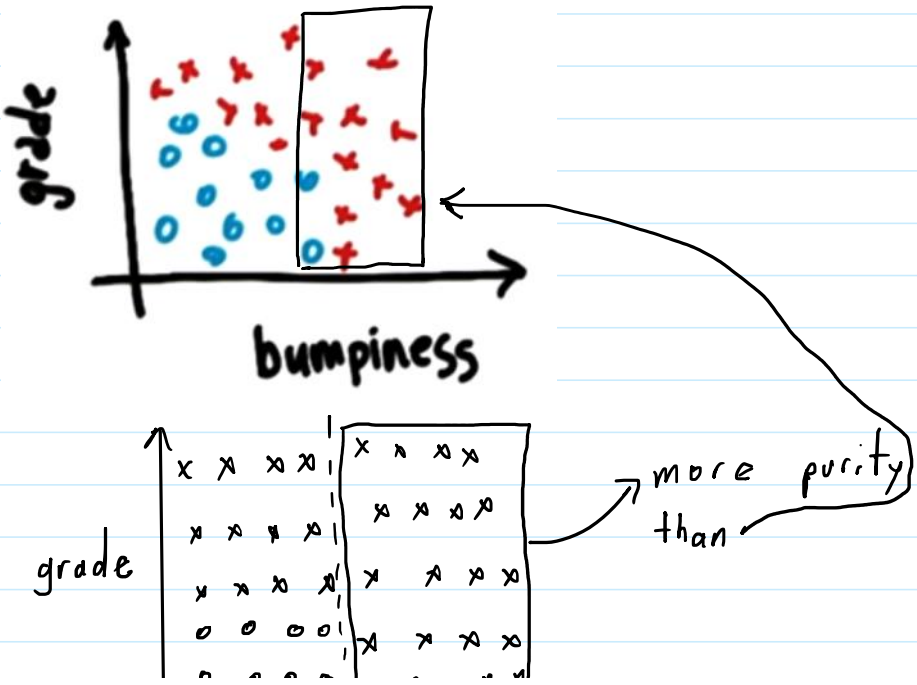


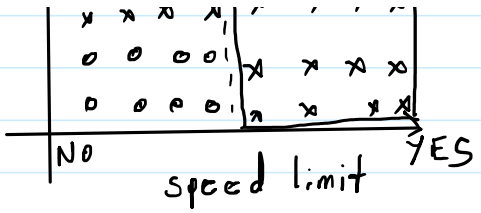
→ Tends to overfit if too low

Entropy → Controls how a DT decides where to split the data

Defn → measure of impurity in a bunch of examples

example | speed limit





Entropy

$$\text{entropy} = \sum_i -p_i \log_2(p_i)$$

p_i: fraction of examples in class i

$\sum_i$: sum over all classes

- all examples are from same class → entropy = 0
- examples are evenly split between classes → entropy = 1.0

Entropy :-

$$\text{entropy} = -\sum_i (p_i) \log_2(p_i)$$

grade	bumpiness	speed limit?	speed
steep	bumpy	yes	slow
steep	smooth	yes	slow
flat	bumpy	no	fast
steep	smooth	no	fast

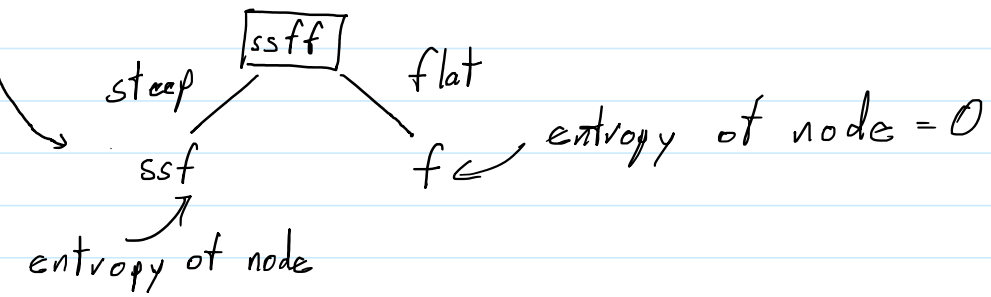
Information gain

Information gain = entropy(parent) − [weighted avg] entropy(children)

Decision Tree algorithm → Maximize information gain

Decision Tree algorithm → Maximize information gain

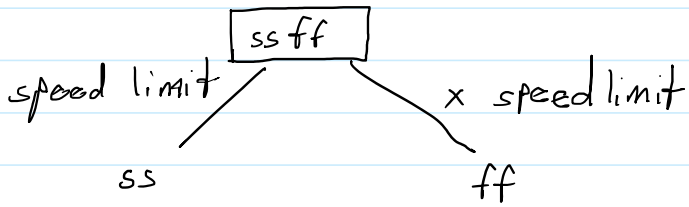
entropy of parent = 1.0



$$\text{entropy (children)} = \frac{3}{4}(0.9184) + \frac{1}{4}(0)$$

$$= 0.6888$$

$$\text{Information gain} = 1 - 0.6888$$
$$= 0.3112$$



Decision Trees Strengths & Weaknesses

Advantages

→ Easy to use

→ Interpretation is easy

Disadvantages

→ Prone to overfitting

→ Build bigger classifiers using something called ensemble methods

Choosing your own adventure

10 October 2021 11:36

Choose your Own Algorithm

→ K-nearest neighbours – classic, simple, easy to understand
→ adaboost } – ensemble methods
→ random forest } meta classifiers built from (usually) decision trees

Dataset and Questions

10 October 2021 14:06

Enron Dataset

Enron Corpus → biggest corpus of open emails

Regression — Salaries & bonuses

→ Clustering (K means)

→ Director } Outlier detection
→ Employee } & removal

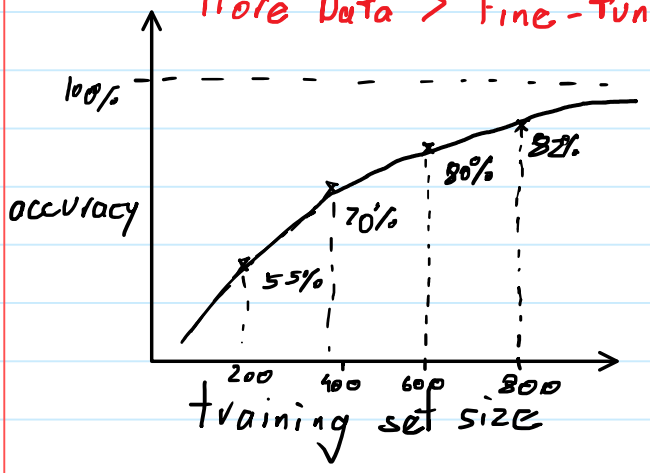
→ bugs

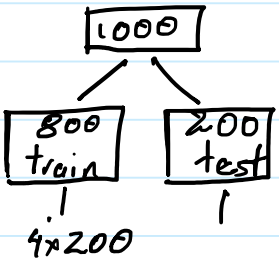
Clustering → Identify people based on movie choices — Netflix

Person of Interest

→ indicted
→ settled without admitting guilt
→ testified in exchange for immunity

More Data > Fine-tuned algorithm





Regression

17 October 2021 18:29

```
from sklearn import linear_model
clf = linear_model.LinearRegression()
clf.fit([[0,0], [1,1], [2,2], [0,1,2]])
clf.coef_
clf.intercept_
```

Unsupervised Learning

24 October 2021 12:25

K means on sklearn

```
from sklearn.cluster import Kmeans
import numpy as np

# Kmeans limitations
# Will output for any fixed training set remain the same? -No
X=np.array([[1,2],[1,4],[1,0], [10,2],[10,4],[10,0]])
kmeans = Kmeans(n_clusters = 2, random_states = 0).fit(X)
kmeans.predict([[0,0],[12,3]])
kmeans.cluster_centers_
```

Feature Scaling

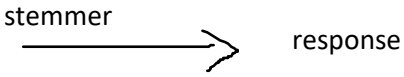
24 October 2021 12:57

```
from sklearn.preprocessing import MinMaxScaler
import numpy
scaler = MinMaxScaler
rescaled_weights = scaler.fit_transform(weights)
```

Text Learning

24 October 2021 13:10

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
bag_of_words = vectorizer.fit(email_list)
bag_of_words = vectorizer.transform(email_list)
print(vectorizer.vocabulary_.get('great'))

nltk.download()
from nltk.corpus import stopwords
sw = stopwords.words('english')
sw[0]

# Not all unique words are different
# unresponsive
# response
# responsivity
# responsiveness
# respond
```

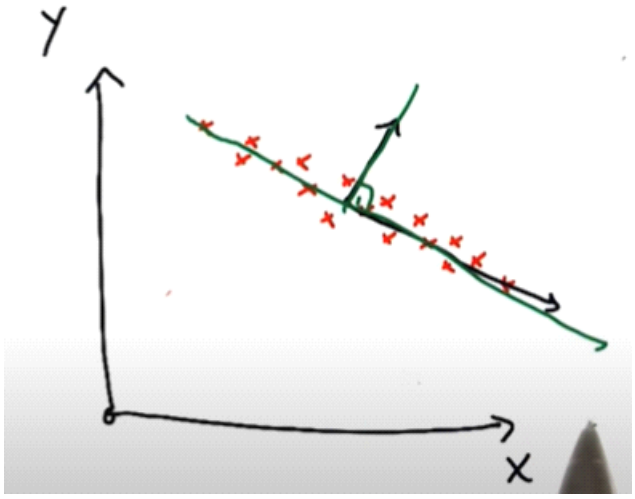


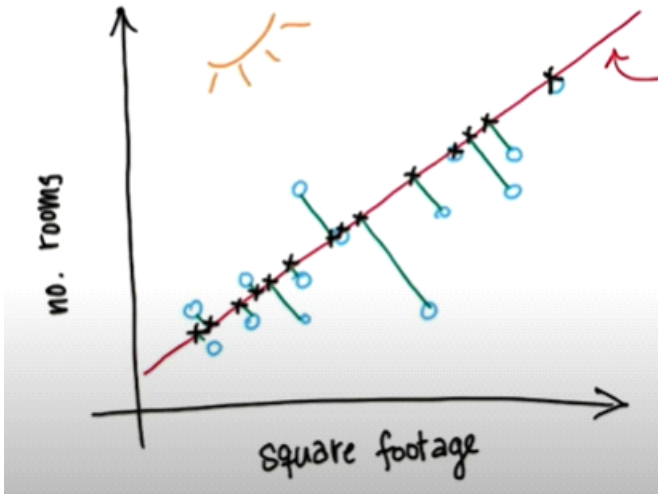

```
Word stemming
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer('english')
stemmer.stem('responsiveness')

stemmer.stem('unresponsive')
```

TfIdf Representation
Tf - Term frequency similar to bag of words
Idf - inverse document frequency weighting by how often word occurs in corpus

Principal Component Analysis

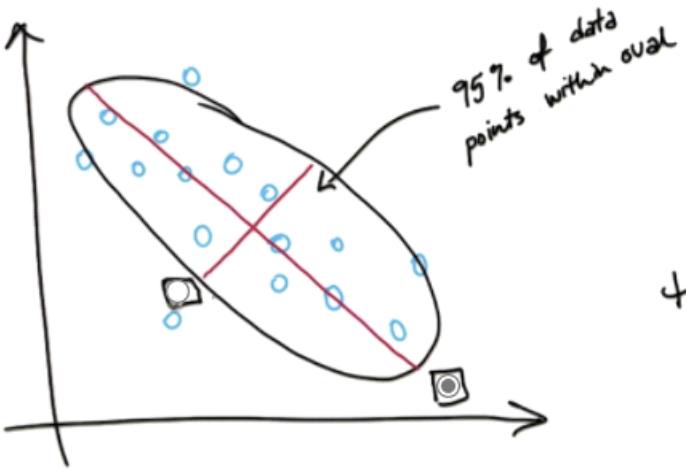
24 October 2021 23:26





How to Determine the Principal Component

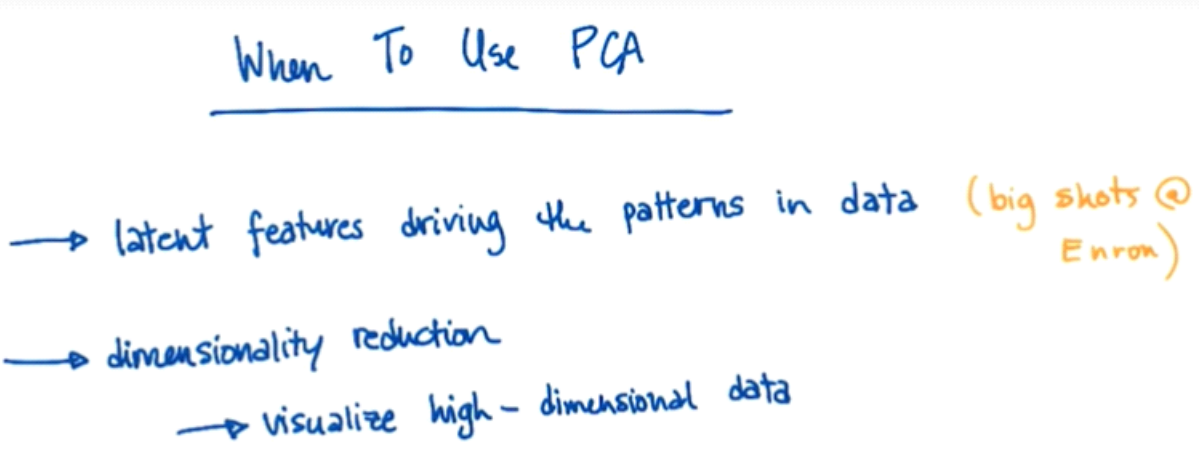
Variance

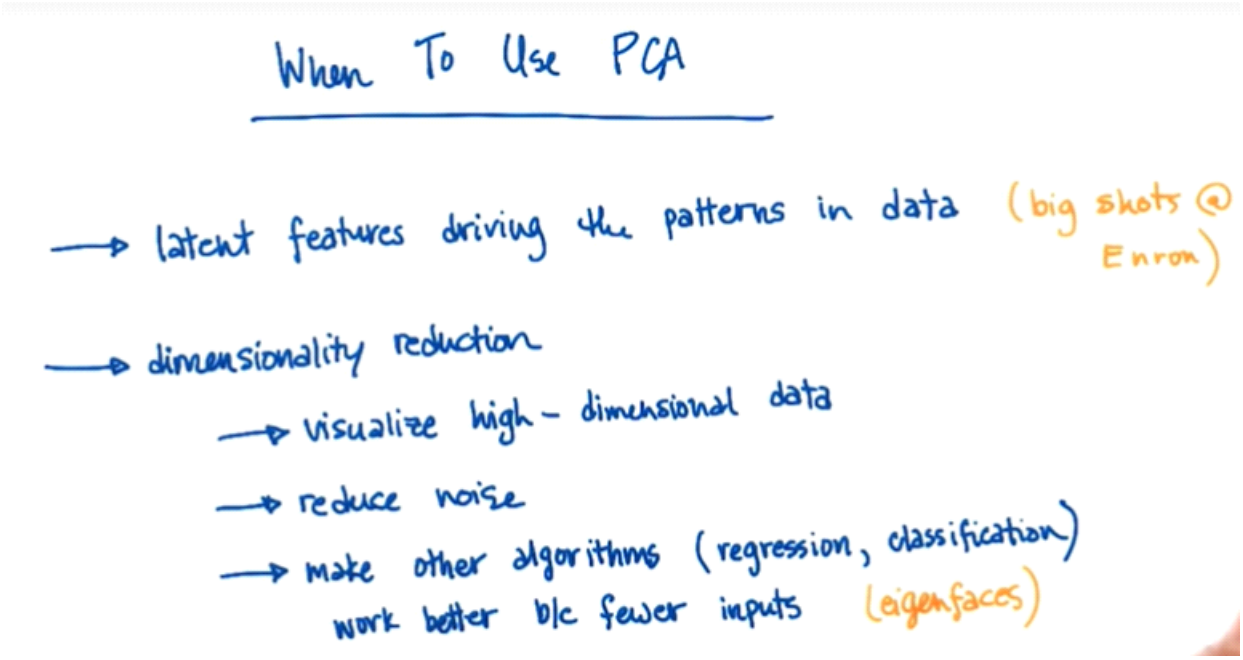


PCA Review

systemized way to transform input features into principal components
use principal components as new features
PCs are directions in data that maximize variance (minimize information loss) when you project/compress down onto them
more variance of data along a PC, higher that PC is ranked
most variance/most information is the first PC
second most variance (without overlapping with first PC) is the second PC
max no of PCs is number of features

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(data)

print(pca.explained_variance_ratio_)
first_pc = pca_components_[0]
second_pc = pca_components_[1]

transformed_data = pca.transform(data)
for ii, jj in zip(transformed_data, data):
        plt.scatter(first_pc[0] * ii[0], first_pc[1] ** ii[0], color = 'r')
        plt.scatter(second_pc[0] * ii[1], second_pc[1] ** ii[1], color = 'c')
        plt.scatter(jj[0], jj[1], color = 'r')
```





PCA for facial recognition

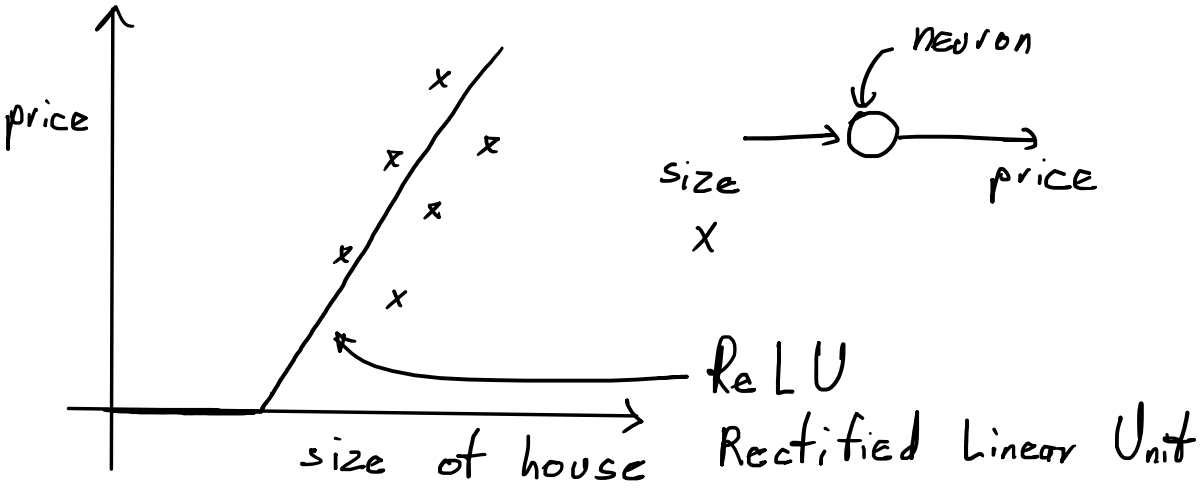
1) pictures of faces generally have high input dimensionality
2) faces have general that could be captures in smaller number of dimensions

```
###############################################################################
# Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
# dataset): unsupervised feature extraction / dimensionality reduction
n_components = 150

print "Extracting the top %d eigenfaces from %d faces" % (n_components, X_train.shape[0])
t0 = time()
pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)
print "done in %0.3fs" % (time() - t0)

eigenfaces = pca.components_.reshape((n_components, h, w))

print "Projecting the input data on the eigenfaces orthonormal basis"
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print "done in %0.3fs" % (time() - t0)
```

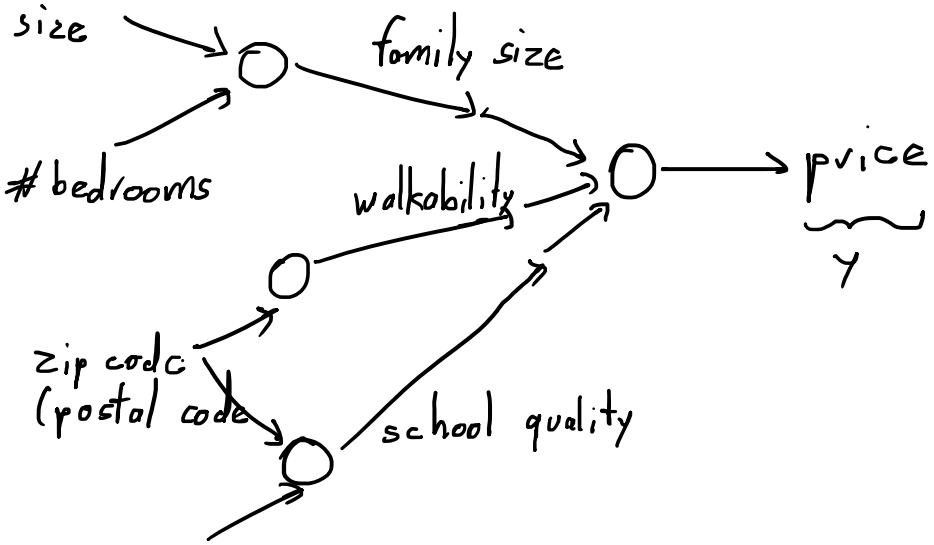
Neural Networks and Deep Learning

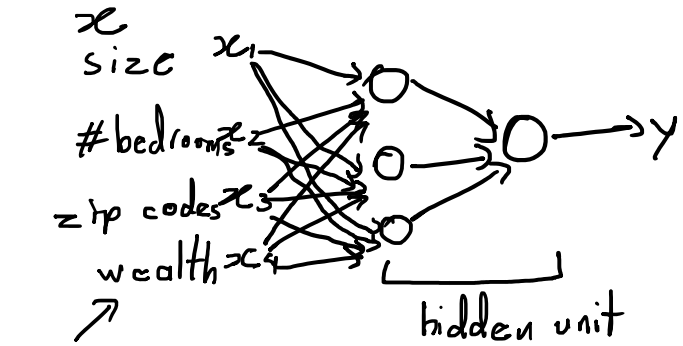
13 September 2021 12:31

What is a neural network



size, #bedrooms, family size





input features
(input layer)

(input layer)

densely connected

Supervised Learning

Input (x)	Output (y)	Application	
Home features	Price	Real Estate	Standard NN
Ad, user info	Click on ad?	Online Ad	
Image	Object (1..1000)	Photo Tagging	CNN
Audio	Text Transcript	Speech Recognition	RNN
English	Chinese	Machine Translation	RNN
Image, Radar info	Position of other cars	Autonomous driving	Custom/ Hybrid

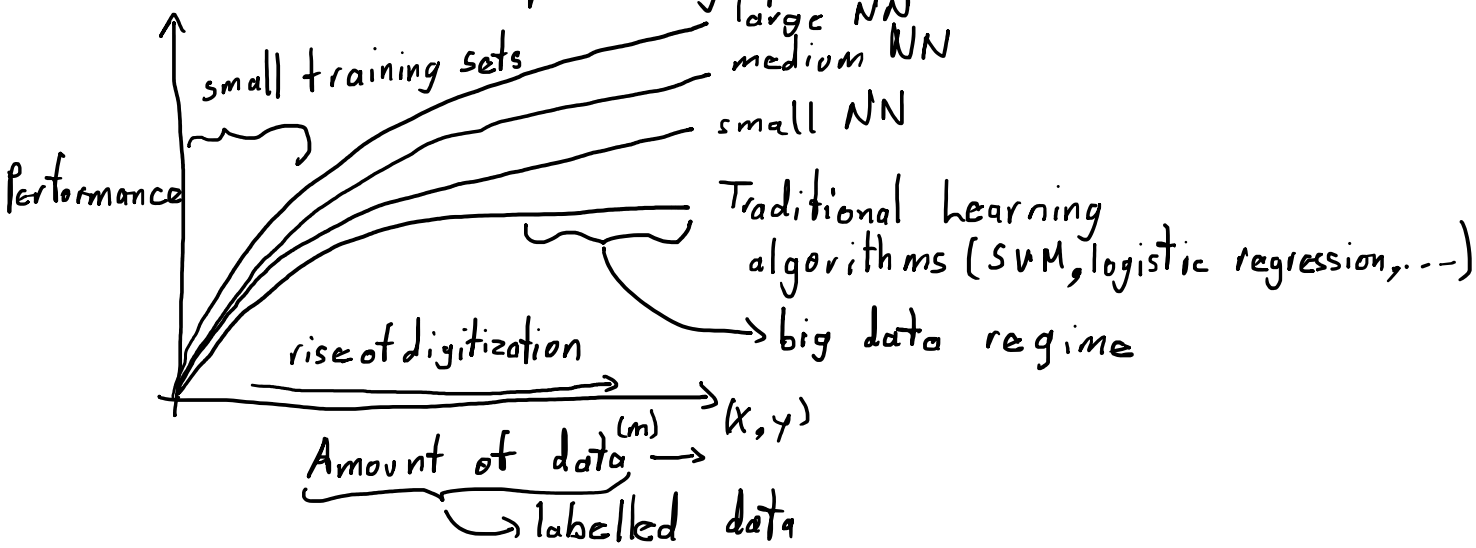
Structured Data

x_1	x_2	x_3	...	Y

Unstructured Data

Audio, Image, Text

Drivers of deep learning process



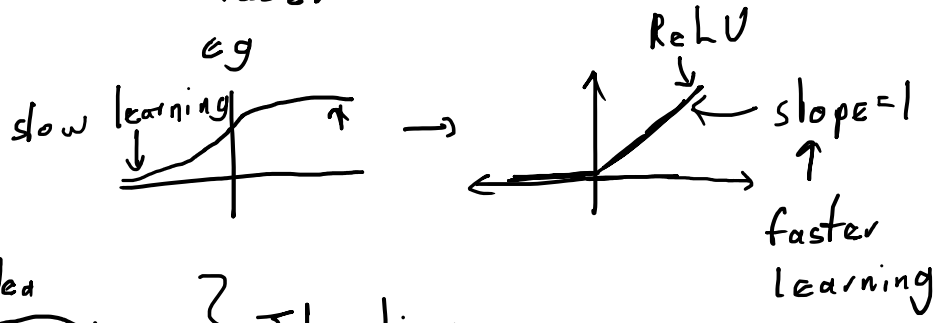
Scale drives deep learning progress

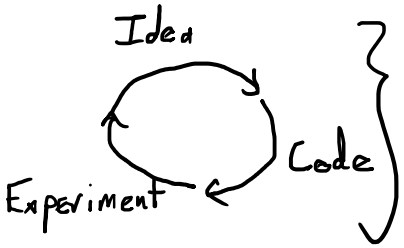
→ data

→ computation

→ algorithms ← recently → trying to make NN faster

eg





Iterative process of discovering NN architecture with better results

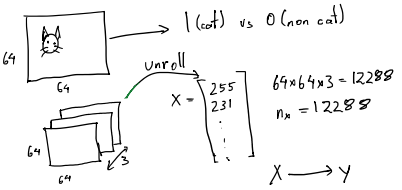
Logistic Regression as a Neural Network

13 September 2021 14:50

Binary Classification

forward propagation →

← backward propagation



(x, y) $x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

m training examples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \vdots, (x^{(m)}, y^{(m)})\}$

$m \rightarrow m_{train}$ m_{test} = #test examples

$$X = \begin{bmatrix} | & | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots & & x^{(m)} \\ | & | & | & | & & | \end{bmatrix}$$

$X \in \mathbb{R}^{n_x \times m}$ $(X.shape = (n_x, m))$

$Y = [\, y^{(1)}, y^{(2)}, y^{(3)}, \cdots y^{(m)}]$

$Y \in \mathbb{R}^{1 \times m}$ $(y.shape = (1, m))$

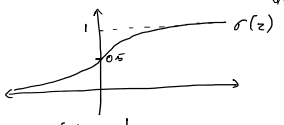
Logistic Regression

Given x, want $\hat{y} = P(y=1|x)$

$x \in \mathbb{R}^{n_x}$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

Output: $\hat{y} = w^T x + b$ ← not bounded! b/w [0, 1]



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

If z is large $\sigma(z) \approx \frac{1}{1+0} = 1$

If z is large negative number

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx 0$$

w & b are kept separate

Logistic Regression Cost Function

$\hat{y} = \sigma(w^T x + b)$

Given $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \cdots (x^{(m)}, y^{(m)})\}$

want $\hat{y}^{(i)} \approx y^{(i)}$

Loss (error) function: $L(\hat{y}, y)$ ← don't use squared error (multiple local minima)

Convex fn

$$L(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$$

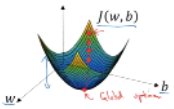
If $y = 1$: $L(\hat{y}, y) = -\log \hat{y}$ ← want $\hat{y}$ large ↳ $\hat{y} = 1$

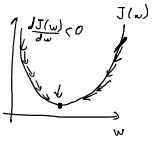
If $y = 0$: $L(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\hat{y}$ small ↳ $\hat{y} = 0$

On entire training set

Cost fn: $J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log y^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right]$$





Repeat {

$w := w - \alpha \frac{dJ(w)}{dw}$

}

dw

$w := w - \alpha\, dw$

$J(w, b)$

Repeat {

$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$ ← partial derivative

$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$

}

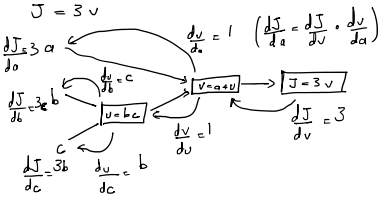
Computation Graph

$J(a, b, c) = 3(a + bc)$

$u = bc$

$v = a + u$

$J = 3v$



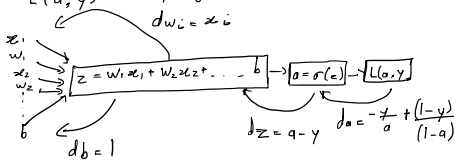
$\frac{d(\text{Final Output Variable})}{d(\text{Var})} \rightarrow d\text{Var}$

Computation Graph

$z = w^T x + b$

$\hat{y} = \sigma(z) = a$

$L(a, y) = -(y \log(a) + (1-y) \log(1-a))$



$a = \frac{e^z}{1 + e^z}$

$\frac{da}{dz} = \frac{e^z(1+e^z) - e^z(e^z)}{(1+e^z)^2}$

$= \frac{e^z}{(1+e^z)} \cdot \frac{e^z}{(1+e^z)} \quad e^{-z}$

$= a^2 e^{-z}$

$\frac{dL}{dz} = a^2 e^{-z}\left(-\frac{y}{a} + \frac{(1-y)}{(1-a)}\right)$

$= -a\,y e^{-z} + \frac{(1-y)(1+e^z)\,e^z}{(1+e^z)^2}$

$= -\frac{y}{1+e^z} - \frac{y\,e^z}{(1+e^z)} + \frac{e^z}{1+e^z}$

$= a - y\frac{(1+e^z)}{(1+e^z)} = a - y$

$dw_i = x_i(a - y)$

$db = a - y$

Gradient Descent on m-training examples

$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(a^{(i)}, y)$

$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$

$dw_1^{(i)}, \; dw_2^{(i)}, \; db^{(2)}$

$\frac{\partial}{\partial w_1} J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_1} L(a^{(i)}, y^{(i)})$

$dw_1^{(i)}$, $dw_2^{(i)}$, $db^{(2)}$

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} L(a^{(i)}, y^{(i)})}_{dw_1^{(i)} \Rightarrow (x^{(i)}, y^{(i)})}$$

$J=0$; $dw_1=0$; $dw_2=0$; $db=0$

for $i=1$ to m

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

Used as accumulators:

$dw_1 += x_1^{(i)} dz^{(i)}$

$dw_2 += x_2^{(i)} dz^{(i)}$ } $n=2$

$db += dz^{(i)}$

$J /= m$

$dw_1 /= m$

$dw_2 /= m$

$db /= m$

$w_1 := w_1 - \alpha\, dw_1$

$w_2 := w_2 - \alpha\, dw_2$

$b := b - \alpha\, db$

Two Weaknesses
- ↳ 2 for loops → for m training examples
- → for n features
  - ↳ Use vectorization to get rid of "for" loops

Vectorization

↳ Enables code to run quickly

$z = w^T X + b$

Non vectorized

```
z = 0
for i in range(n_x):
    z += w[i] * x[i]
z += b
```

Vectorized

```
z = np.dot(w, x) + b
```

$w^T x$

↖ faster

```
import numpy as np
a = np.array([1,2,3,4])
print(a)
import time
a = np.random.rand(100000)
b = np.random.rand(100000)
tic = time.time()
c = np.dot(a,b)
toc = time.time()
print(f"Vectorized implementation: {toc-tic}s")

c = 0
tic = time.time()
for i in range(100000):
    c += a[i] * b[i]
toc = time.time()
print(c)
print(f"For loop: {toc-tic}s")
```

↖ Take much much more time

GPU } SIMD
CPU } ↳ single instruction multiple data

Whenever possible avoid explicit for loops

$v = Av$

$v_i = \sum A_{ij} v_j$

Whenever possible avoid explicit for loops

$u = Av$

$u_i = \sum_j A_{ij} v_j$

```
u = np.zeros((n,1))
for i ....
    for j ....
        u[i] += A[i][j] * v[j]
```

```
u = np.dot(A, v)
```

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \qquad u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

```
u = np.zeros((n, 1))
for i in range(n):
    u[i] = math.exp(v[i])
```

```
import numpy as np
u = np.exp(v)
```

np.log, np.abs, np.max, v**2, 1/v → element-wise

$J = 0;\ dw_1 = 0;\ dw_2 = 0;\ db = 0$

for i = 1 to m

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

Used as accumulators:

$dw_1 += x_1^{(i)} dz^{(i)}$

$dw_2 += x_2^{(i)} dz^{(i)}$

$db += dz^{(i)}$

n = 2

for j = 1 ... nx

$dw_j += \ldots$

$J /= m$

$dw_1 /= m$

$dw_2 /= m$

$db /= m$

$w_1 := w_1 - \alpha\, dw_1$

$w_2 := w_2 - \alpha\, dw_2$

$b := b - \alpha\, db$

$J = 0,\ dw = np.zeros((n_x, 1)),$

$db = 0$

for i = 1 to m

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J += -[y^{(i)} \log(a^{(i)}) + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw += x^{(i)} dz^{(i)}$

$J = J/m,\ db = db/m,\ dw /= m$

np.dot(w.T, x) ↓ even better implementation

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$Z = (w^T X) + b$ ← broadcast

vectorized implementation of fn

$A = \text{sigmoid}(Z)$

$J = -[np.dot(y, np.log(a)) + np.dot(1-y, np.log(1-a))]/m$

$dz = a - y$

$dw = \frac{1}{m}\, np.dot(X, dz.T)$

$db = \frac{1}{m}\, np.sum(dz)$

To conclude :-

$Z = w^T X + b$

$= np.dot(w.T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

$dw = \frac{1}{m} X dZ^T$

$db = \frac{1}{m}\, np.sum(dZ)$

$w := w - \alpha\, dw$

$b := b - \alpha\, db$

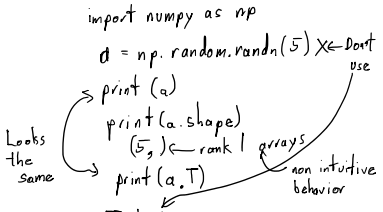
} Iterations still require a for loop

Broadcasting example

```
A = np.random.rand(16).reshape(4,4)
cal = A.sum(axis=0)
percentage = 100*A/cal.reshape(1,4)
```

0 ↓ 1

Eliminate bugs



Instead use

```
a = np.random.randn(5,1)
assert(a.shape == (5,1))
a = a.reshape((5,1))
```

Logistic Regression Cost Function

$\hat{y} = \sigma(w^T x + b)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$

If $y=1$: $p(y|x) = \hat{y}$

If $y=0$: $p(y|x) = 1-\hat{y}$

$p(y|x) = \hat{y}^{y}(1-\hat{y})^{(1-y)}$

$\log(p(y|x)) = y\log\hat{y} + (1-y)\log(1-\hat{y})$

$= -L(\hat{y}, y)$ ← minimize loss fn → maximize probability

$p(\text{labels in training set}) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)})$

$\log(p(\text{labels in training set})) = \sum_{i=1}^{m} \log(p(y^{(i)}|x^{(i)})$

$= \sum L(\hat{y}, y)$

$= \left(\frac{1}{m}\right) * -\sum L(\hat{y}, y)$

Scaling factor

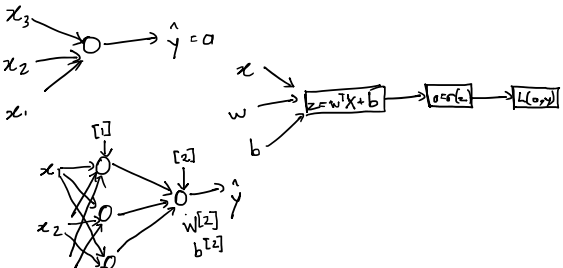
Code:-

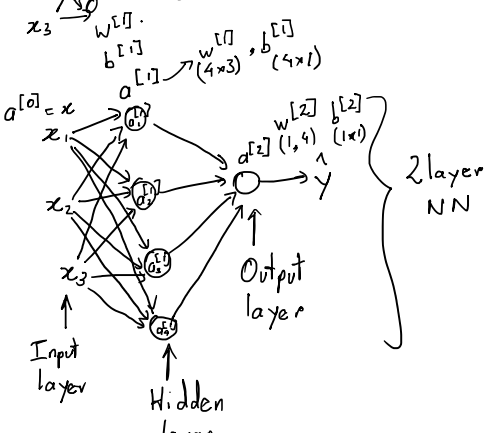
```
import numpy as np
# Load data
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y = load_dataset()

# Flatten
train_set_x_flatten = train_set_x_orig.reshape(train_set_x_orig.shape[0],-1)
test_set_x_flatten = test_set_x_orig.reshape(test_set_x_orig.shape[0],-1)

# Normalize
train_set_x_flatten = train_set_x_flatten/255
test_set_x_flatten = test_set_x_flatten/255

# sigmoid function
def sigmoid(z):
    s = 1/(1+np.exp(-z))
    return s

# Initialise parameters
def initialize_with_zeros(dim):
    w = np.zeros((dim,1))
    b = 0 #will be broadcasted to appropriate shape
    assert(w.shape ==(dim,1))
    assert(isinstance(b,float) or isinstance(b,int))
    return w, b

# Compute cost function and gradient
def propagate(w, b, X, y):
    m = X.shape[1]
    # Forward Propagation
    z = np.dot(w.T, X) + b
    A = sigmoid(z)
    cost = -(np.dot(y, np.log(A).T) + np.dot(1 - y, np.log(1-A).T))/m
    # Backward Propagation
    dz = A - Y
    dw = np.dot(X, dz.T)/m
    db = np.sum(dz)/m

    assert(dw.shape == w.shape)
    assert(db.dtype == float)
    cost = np.squeeze(cost)
    assert(cost.shape == ())

    grads = {"dw": dw, "db": db}
    return cost, grads

# Update parameters using gradient descent
def optimize(w, b, X, y, num_iterations, learning_rate, print_cost =False):
    costs = []
    for i in range(num_iterations):
        # Gradients and cost computation
        gradients, cost = propagate(w, b, X, y)
        dw = gradients["dw"]
        db = gradients["db"]

        # Update parameters
        w = w - (learning_rate * dw)
        b = b - (learning_rate * db)

        if i % 100 ==0:
            costs.append(cost)

        if print_cost and i % 100 == 0:
            print(cost)

        params = {"w": w, "b":b}
        grads = {"dw": dw, "db": db}

        return params, costs, grads

# Lastly define the predict function to be used on the test set
def predict(w, b, X):
    z = np.dot(w.T, X) + b
    A = sigmoid(z)

    Y_prediction = np.zeros((1, X.shape[1]))
    for i in range(X.shape[1]):
        if A[i] >= 0.5:
            Y_prediction[i] = 1
        else:
            pass
    assert(Y.shape == (1,m))
    return Y_prediction

# Merge all functions into a model
def model(X_train, y_train, X_test, y_test, num_iterations = 2000, learning_rate = 0.01, print_cost =
False):
    w, b = initialize_with_zeros(X_train.shape[0])
```

```
parameters, costs, grads = optimise(w, b, X_train, y_train, learning_rate, num_iterations,
print_cost)
w = parameters["w"]
b = parameters["b"]

Y_prediction_train = predict(w, b, X_train)
Y_prediction_test = predict(w, b, X_test)

training_accuracy = 1 - np.mean(np.abs(Y_prediction_train - y_train))
test_accuracy = 1 - np.mean(np.abs(Y_prediction_test - y_test))

print(f"Train accuracy: {100*training_accuracy}")
print(f"Test accuracy: {100*test_accuracy}")

d ={"costs":costs, "Y_prediction_train": Y_prediction_train, "Y_prediction_test":
Y_prediction_test, "w":w, "b", b, "learning_rate": learning_rate, "num_iterations":
num_iterations}

return d
```

Shallow Neural Networks

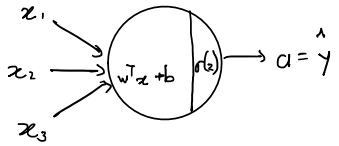
14 September 2021 15:01





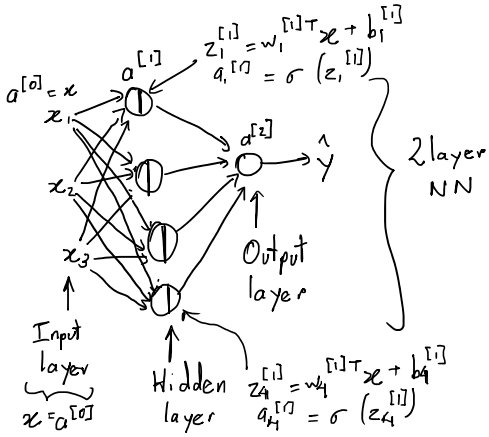
$$a^{[0]} = x$$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_4^{[1]} \end{bmatrix}$$



$$z = w^T x + b$$

$$a = \sigma(z)$$



$$x = a^{[0]}$$

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}$$

$$\begin{bmatrix} — & w_1^{[1]T} & — \\ — & w_2^{[1]T} & — \\ — & w_3^{[1]T} & — \\ — & w_4^{[1]T} & — \end{bmatrix}_{4\times 3}$$

$$\begin{bmatrix} — & w_1^{[1]T} & — \\ — & w_2^{[1]T} & — \\ — & w_3^{[1]T} & — \\ — & w_4^{[1]T} & — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} \Rightarrow \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}$$

$$W^{[1]} \qquad b^{[1]}$$

$W^{[1]}$ $b^{[1]}$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

Given input x $\rightsquigarrow a^{[0]}$

$z^{[1]}_{(4\times1)} = W^{[1]}_{(4\times3)} a^{[0]}_{(3\times1)} + b^{[1]}_{(4\times1)}$

$a^{[1]}_{(4\times1)} = \sigma(z^{[1]})_{(4\times1)}$

$z^{[2]}_{(1\times1)} = W^{[2]}_{(1\times4)} a^{[1]}_{(4\times1)} + b^{[2]}_{(1,1)}$

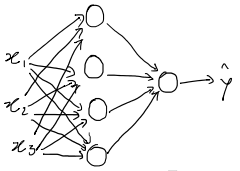
$a^{[2]} = \sigma(z^{[2]})$

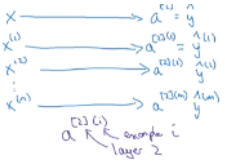
$\hat{y}$ $\nearrow_{(1,1)}$ $(1,1)$

Dimensions of $w \rightarrow$ (number of units in current layer, number of input features (n_x))

Dimensions of $b \rightarrow$ (number of units in current layer, 1)

Vectorizing across multiple examples





$z^{[1]} = W^{[1]} x + b^{[1]}$

$a^{[1]} = \sigma(z^{[1]})$

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

$a^{[2]} = \sigma(z^{[2]})$

for $i = 1$ to m

$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$

$a^{[1](i)} = \sigma(z^{[1](i)})$

$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$

$a^{[2](i)} = \sigma(z^{[2](i)})$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

n_x, m

Vectorized implementation:-

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$A^{[1]} = \sigma(Z^{[1]})$

$Z^{[2]} = W^{[2]} X + b^{[2]}$

$A^{[2]} = \sigma(Z^{[2]})$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

training examples $\leftrightarrow$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix} \downarrow \text{\# hidden units}$$

$A^{[i]} = \begin{bmatrix} | & | & & | \\ a^{[i](1)} & a^{[i](2)} & \cdots & a^{[i](m)} \\ | & | & & | \end{bmatrix}$ ↓ # hidden units

← # training examples

Justification for vectorised implementation

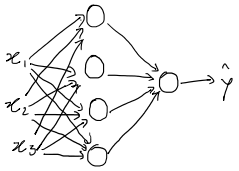
$z^{[i](1)} = W^{[i]} X^{(1)} + b^{[i]}$ → 0 $\quad Z^{[i](2)} = W^{[i]} X^{(2)} + b^{[i]}$ → 0

$W^{[i]} = \begin{bmatrix} - w_1^{[i]T} - \\ - w_2^{[i]T} - \\ \vdots \end{bmatrix} \quad W^{[i]} X^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \quad W^{[i]} X^{(2)} = \begin{bmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$

$W^{[i]} \begin{bmatrix} | & | & & | \\ X^{(1)} & X^{(2)} & \cdots & X^{(m)} \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & & \\ \cdot & \cdot & & \\ \cdot & \cdot & & \\ \cdot & \cdot & & \end{bmatrix} = \begin{bmatrix} | & | & & \\ z^{[i](1)} & z^{[i](2)} & \cdots & \\ | & | & & \end{bmatrix}$

$Z^{[i]}$

Activation Functions



$Z^{[1]} = W^{[1]} X + b^{[1]}$

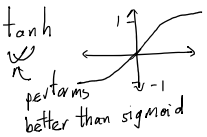
$A^{[1]} = g(z^{[1]})$

$Z^{[2]} = W^{[2]} X + b^{[2]}$

$A^{[2]} = g(z^{[2]})$

- sigmoid ← best for o/p layer

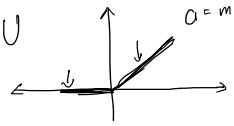
- tanh → $\frac{e^z - e^{-z}}{e^z + e^{-z}}$



performs better than sigmoid

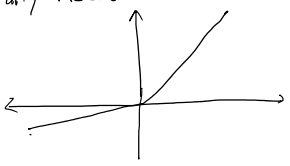
if $z \to \infty$ or $z \to -\infty$ $\frac{da}{dz} = 0$ ↑ gradient descent is slow

- ReLU → $a = \max(0, z)$



← when in doubt use ReLU

- Leaky ReLU



$a = \max(0.01z, z)$

Why do we need non-linear activation fn?

$a^{[1]} = z^{[1]} = W^{[1]} X + b^{[1]}$

$a^{[2]} = z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

$a^{[2]} = W^{[2]} (W^{[1]} X + b^{[1]}) + b^{[2]}$

$= W' X + b'$

Linear activation fn used only when output takes on IR values ie $y \in \mathbb{R}$ (housing prices)

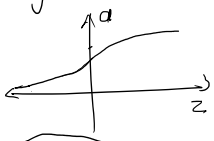
↳ HU can use ReLU, tanh etc

↑ housing prices

↳ > 0

Derivatives of activation fn

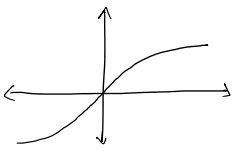
Sigmoid



$$g(z) = \frac{1}{1+e^{-z}}$$

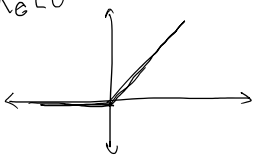
$$g'(z) \rightarrow \frac{d}{dz} g(z) = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$
$$= g(z)(1-g(z))$$
$$= a(1-a)$$

tanh



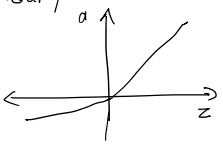
$$g'(z) = 1-(\tanh(z))^2$$
$$= 1-a^2$$

ReLU



$$g(z) = \max(0, z)$$
$$g'(z) = \begin{cases} 0 & z<0 \\ 1 & z \geq 0 \end{cases}$$
~~undefined $z=0$~~

Leaky ReLU



$$g(z) = \max(0.01z, z)$$
$$g'(z) = \begin{cases} 0.01 & z<0 \\ 1 & z \geq 0 \end{cases}$$

Gradient descent for NN

Params: $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$

$(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$

Cost fn: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$

Repeat {

Compute pred^n ($\hat{y}^{(i)}$, $i=1 \ldots m$)

$dW^{[1]} = \frac{dJ}{dW^{[1]}}$, $db = \frac{dJ}{db^{[1]}}$

$W^{[1]} = W^{[1]} - \alpha\, dW^{[1]}$

$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$

$$W^{[2]} = W^{[2]} - \alpha\, dW^{[2]}$$
$$b^{[2]} = b^{[2]} - \alpha\, db^{[2]}$$

Forward Propagation

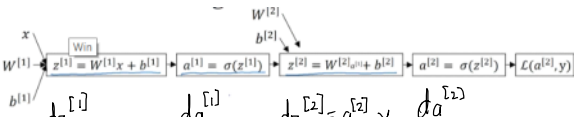
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$

Backpropagation

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis=1, keepdims=True)$$
$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * g^{[1]'}(Z^{[1]}) \quad (n^{[1]}, m)$$

(* : element-wise product)

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$
$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis=1, keepdims=True) \quad (n^{[1]}, 1)$$



$dz^{[1]}$ = $W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[2]})$

$da^{[1]}$

$dz^{[2]} = a^{[2]} - y$ $\quad da^{[2]}$

$dW^{[2]} = dz^{[2]} a^{[1]T}$

$db = dz^{[2]}$

$dW^{[1]} = dz^{[1]} \cdot X^T$

$db^{[1]} = dz^{[1]}$

6 key equations:

$$dz^{[2]} = a^{[2]} - y$$
$$dW^{[2]} = dz^{[2]} a^{[1]T}$$
$$db^{[2]} = dz^{[2]}$$
$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$$
$$dW^{[1]} = dz^{[1]} x^T$$
$$db^{[1]} = dz^{[1]}$$

Vectorize $\Longrightarrow$

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis=1, keepdims=True)$$
$$dZ^{[1]}_{(n^{[1]}, m)} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$
$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis=1, keepdims=True)$$

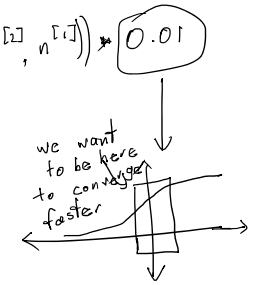
Random Initialization

$W^{[1]} = np.random.randn((n^{[2]}, n^{[1]})) * 0.01$

$b^{[1]} = np.zeros((n^{[2]}, 1))$

$W^{[2]} =$

$b^{[2]} =$



Implementation Example :-

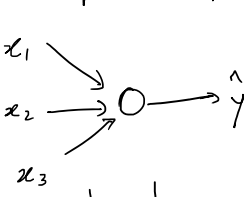
```
import numpy as np
import matplotlib.pyplot as plt
from testCases_v2 import *
import sklearn
import sklearn.datasets
import sklearn.linear_model

%matplotlib inline

np.random.seed(1)
```

```
X,Y = load_planar_dataset()
# Simple logistic regression classifier
clf = sklearn.linear_model.LogisticRegressionCV()
clf.fit(X.T,Y.T)

print(f'Accuracy of logistic regression: {(np.dot(Y,LR_predictions) + np.dot(1-Y,1-LR_predictions))/float(Y.size)*100}%')
# Accuracy is 47%


# Define layer sizes

def layer_sizes(X,Y):
    n_x = X.shape[0]
    n_h = 4
    n_y = Y.shape[0]
    return (n_x, n_h, n_y)

def initialize_parameters(n_x, n_h, n_y):
    W1 = np.random.randn((n_h, n_x))*0.01
    b1 = np.zeros((n_h,1))
    W2 = np.random.randn((n_y, n_h))*0.01
    b2 = np.zeros((n_y,1))

    assert(W1.shape == (n_h,n_x))
    assert(b1.shape == (n_h, 1))
    assert(W2.shape == (n_y,n_h))
    assert(b2.shape == (n_y,  1))

    parameters = {"W1":W1, "b1": b1, "W2":W2, "b2":b2}

    return parameters

def forward propagation(X, parameters):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = {"Z1":Z1, "A1":A1, "Z2":Z2, "A2":A2}

    return cache

def compute_cost(A2, Y, parameters):
    m = Y.shape[1]
    cost = -np.sum((np.dot(Y, np.log(A2).T) + np.dot((1-Y), np.log((1-A2)).T)))/m
    cost = np.squeeze(cost)
    assert(isinstance(cost, float))
    return cost

def backward_propagation(parameters, cache, X,Y):

    m = X.shape[1]

    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    Z1 = cache["Z1"]
    A1 = cache["A1"]
    Z2 = cache["Z2"]
    A2 = cache["A2"]

    dZ2 = A2 - Y
    dW2 = np.dot(dZ2, A1.T)/m
    db2 = np.sum(dZ2, axis = 1, keepdims = True)

    dZ1 = np.dot(W2.T, dZ2) * (1-A1**2)
    dW1 = np.dot(dZ2, X.T)/m
    db1 = np.sum(dZ1, axis = 1, keepdims = True)/m

    grads = {"dW1":dW1, "db1":db1, "dW2":dW2, "db2":db2}
    return grads

def update_parameters(parameters, grads, learning_rate = 1.2):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    dW1 = grads["dW1"]
    db1 = grads["db1"]
    dW2 = grads["dW2"]
    db2 = grads["db2"]

    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1
    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2

    parameters = {"W1":W1, "b1": b1, "W2":W2, "b2":b2}

    return parameters

def nn_model(X,Y, n_h, learning_rate, num_iterations = 10000, print_cost = False):
    n_x, n_h, n_y = layer_sizes(X, Y)
    parameters = initialize_parameters(n_x, n_h, n_y)

    for i in range(num_iterations):
        cache = forward_propagation(X, parameters)
        cost = compute_cost(cache["A2"], Y, parameters)
        grads = backward_propagation(parameters, cache, X, Y)
        parameters = update_parameters(parameters, grads, learning_rate)
        if print_cost and i%1000 ==0:
            print(f'Cost after {i} : {cost}')

    return parameters

def predict(X, parameters):
    cache = forward_propagation(X, parameters)
    predictions = cache["A2"] > 0.5
    return predictions

predictions = predict(parameters, X)

print(f'Accuracy of neural network: {(np.dot(Y,predictions) + np.dot(1-Y,1-predictions))/float(Y.size)*
100}%')
# Accuracy is 90%
```

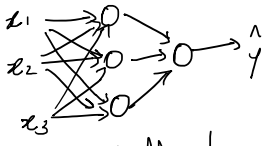
Deep Neural Networks

15 September 2021 13:11

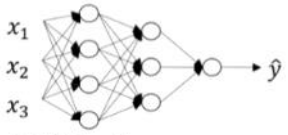
Deep L-Layer neural network



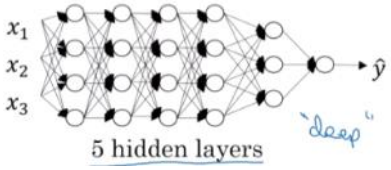
logistic regression



1 hidden layer



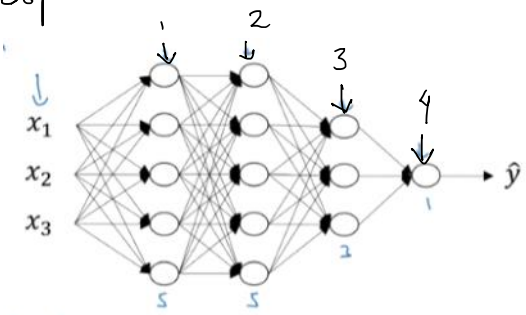
2 hidden layers



5 hidden layers

View number of hidden layers as another hyperparameter

Deep NN notation



$n^{[1]} = 5$

$n^{[2]} = 5$

$n^{[3]} = 3$

$n^{[4]} = 1$

$n^{[0]} = n_x = 3$

$a^{[l]}$ → activations in layer l

$a^{[l]} = g^{[l]}(z^{[l]})$, $w^{[l]}$ ⇒ weights for $z^{[l]}$

Forward Propagation

$X: z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = g^{[2]}(z^{[2]})$

⋮

$a^{[4]} = g^{[4]}(z^{[4]})$

In general

$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

$z^{[4]} = W^{[4]} a^{[3]} + b^{[4]}$

$a^{[4]} = g^{[4]}(z^{[4]})$

Vectorized

$z^{[1]} = W^{[1]} X + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

for loop present

$\hat{y} = g^{[4]}(Z^{[4]}) = A^{[4]}$

Getting Matrix Dimensions Right

$z^{[1]} = W^{[1]} X + b^{[1]}$ → $(n^{[1]}, 1)$

$(n^{[1]}, 1) \quad (n^{[1]}, n^{[0]})(n^{[0]}, 1)$

$W^{[2]} = (n^{[2]}, n^{[1]})$

In general $W^{[l]} = (n^{[l]}, n^{[l-1]})$

$b^{[l]} = (n^{[l]}, 1)$

$dW^{[l]}.shape == W^{[l]}.shape$

$db^{[l]}.shape == b^{[l]}.shape$

broadcasted

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$(n^{[1]}, m) = (n^{[1]}, n^{[0]})(n^{[0]}, m)$ $\quad (n^{[1]}, 1)$

Forward & Backward

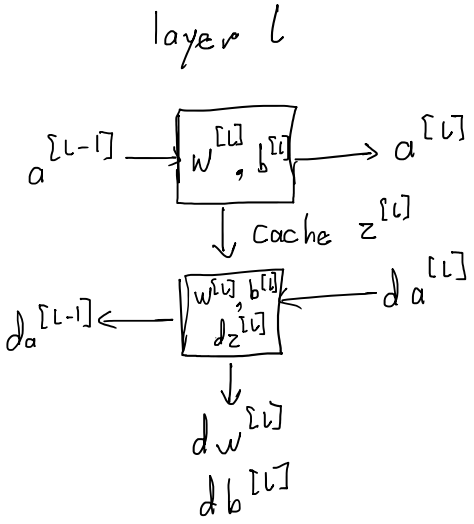
Forward: Input $a^{[l-1]}$, o/p $a^{[l]}$

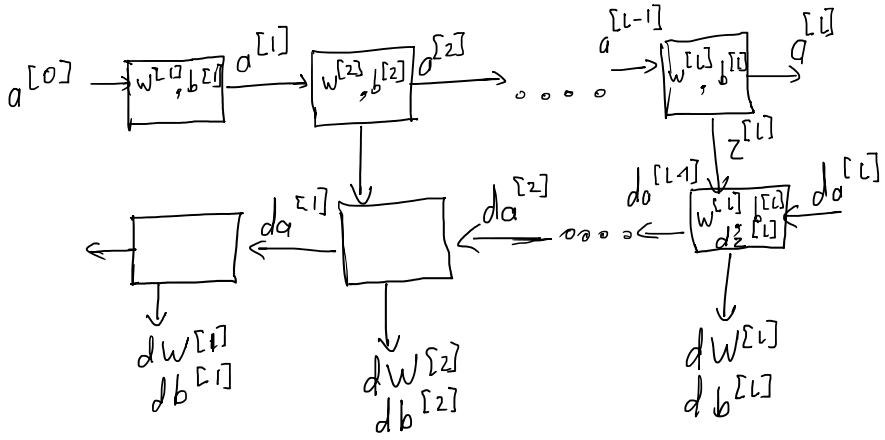
$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ } Cache $z^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

Backward: Input $da^{[l]}$, output $da^{[l-1]}$, $dw^{[l-1]}$, $db^{[l-1]}$

Cache $(z^{[l]})$





Forward propagation for layer l

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

Backward propagation for layer l

→ Input $da^{[l]}$

→ Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$

$$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

⇓ Vectorized

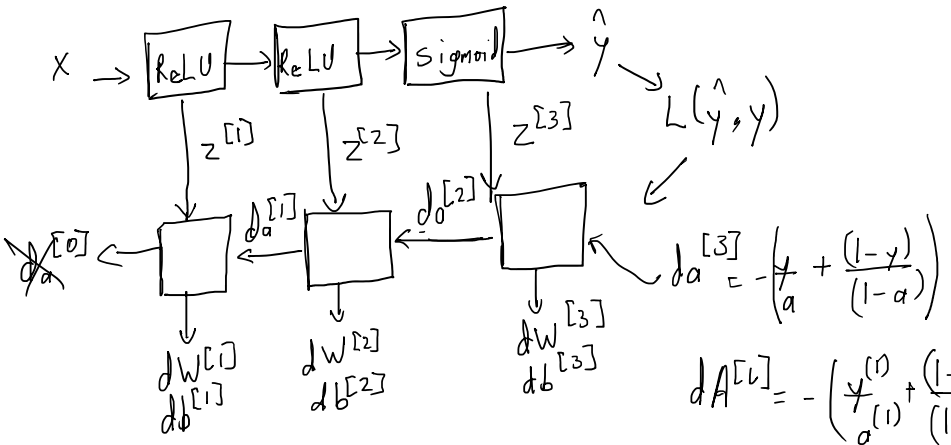
$$dZ^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]}$$

$$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis=1, keepdims=True)$$

$$db^{[L]} = \frac{1}{m} np.sum(dZ^{[L]}, axis=1, keepdims=True)$$

$$dA^{[L-1]} = W^{[L]T} \cdot dZ^{[L]}$$



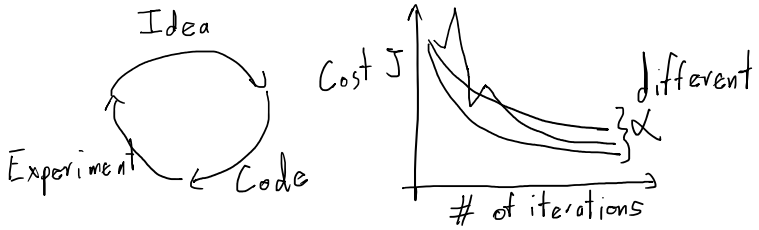
$$dA^{[L]} = -\left(\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} + \cdots \frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})}\right)$$

Parameters vs Hyperparameters

Parameters → $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \ldots$

Hyperparameters: Learning rate α
iterations
hidden layers L
hidden units $n^{[1]}, n^{[2]}$
Choice of activation fn

Later ⇒ Momentum, minibatch size, regularization



Implementation

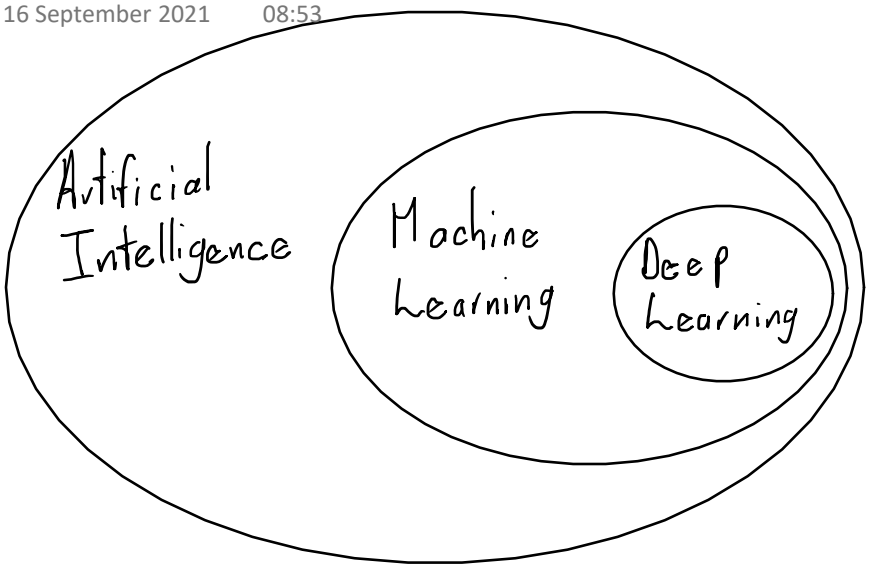
```
import numpy as np
import h5py
import matplotlib.pyplot as plt

%matplotlib inline

np.random.seed(1)

def initialize_parameters_deep(layer_dims):
    np.random.seed(3)
    parameters = {}
    L = len(layer_dims)
```

```
        for i in range(1, L):
            parameters["W"+str(i)] = np.random.randn(layer_dims[i], layer_dims[i-1])*0.01
            parameters["b"+str(i)] = np.random.randn(layer_dims[i], 1)*0.01

        return parameters

def linear_forward(A, W, b):
        Z = np.dot(W, A) + b
        cache = (A, W, b)
        return Z, cache

def_linear_activation_forward(A_prev, W, b, activation):
        if activation == "relu":
            Z, linear_cache = linear_forward(A_prev, W, b)
            A, activation_cache = relu(Z)

        elif activation == "sigmoid":
            Z, linear_cache = linear_forward(A_prev, W, b)
            A, activation_cache = sigmoid(Z)

        cache = (linear_cache, activation_cache)
        return A, cache

def L_model_forward(X, parameters):
        caches = []
        A = X
        L = len(parameters)

        for i in range(1, L):
            A_prev = A
            A, cache = linear_activation_forward(A_prev, parameters["W"+str(i)], parameters["b"+str(i)])
            caches.append(cache)

        AL, cache = linear_activation_forward(A, parameters["W"+str(L)], parameters["b"+str(L)])

        caches.append(cache)

        return AL, caches

def compute_cost(AL, Y):
        m = Y.shape[1]

        cost = -np.sum(np.dot(Y, np.log(AL).T) + np.dot(1-Y, np.log(1-AL).T) )/m
        cost = np.squeeze(cost)
        return cost

def linear_backward(dZ, cache):
        A_prev, W, b = cache
        m = A_prev.shape[1]

        dW = np.dot(dZ, A_prev.T)/m
        db = np.sum(dZ, axis =1, keepdims = True)/m
        dA_prev = np.dot(W.T, dZ)

        return dA_prev, dW, db

def linear_activation_backward(dA, cache, activation):
        linear_cache, activation_cache = cache
        if activation == "relu":
            dZ = relu_backward(dA, activation_cache)
            dA_prev, dW, db = linear_backward(dZ, linear_cache)

        elif activation == "sigmoid":
            dZ = sigmoid_backward(dA, activation_cache)
            dA_prev, dW, db = linear_backward(dZ, linear_cache)

        return dA_prev, dW, db

def L_model_backward(AL, Y, caches):
    grads = {}
    L = len(caches)
    m = AL.shape[1]
    Y = Y.reshape(AL.shape)

   dAL = - (np.divide(Y, AL) - np.divide(1 - Y, 1 - AL))

    current_cache = caches[L-1]
    grads["dA" + str(L-1)], grads["dW" + str(L)], grads["db" + str(L)] = linear_activation_backward(dAL, current_cache, 'sigmoid')

    for l in reversed(range(L-1)):
        current_cache = caches[l]
        dA_prev_temp, dW_temp, db_temp = linear_activation_backward(grads['dA'+str(l+1)], current_cache, 'relu')
        grads["dA" + str(l)] = dA_prev_temp
        grads["dW" + str(l + 1)] = dW_temp
        grads["db" + str(l + 1)] = db_temp

    return grads
```

```
def update_parameters(parameters, grads, learning_rate):
    L = len(parameters) // 2

    for l in range(L):
        parameters["W" + str(l+1)] = parameters["W" + str(l+1)]-((learning_rate)*(grads["dW" + str(l+1)]))
        parameters["b" + str(l+1)] = parameters["b" + str(l+1)]-((learning_rate)*(grads["db" + str(l+1)]))

    return parameters
```

MIT Deep Learning

16 September 2021 08:53



Why deep learning?

Hand engineered features are time-consuming, brittle and not scalable in practice

Can we learn underlying features directly from data

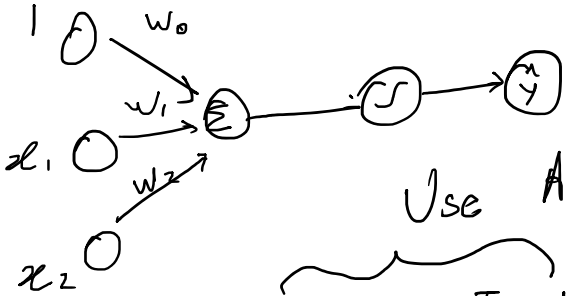
Why Now?

→ Big Data

→ Hardware

→ Software

Perceptron → Forward Propagation



Use Andrew NG's method way way better

$$\hat{y} = \sigma(w_0 + X^T W)$$

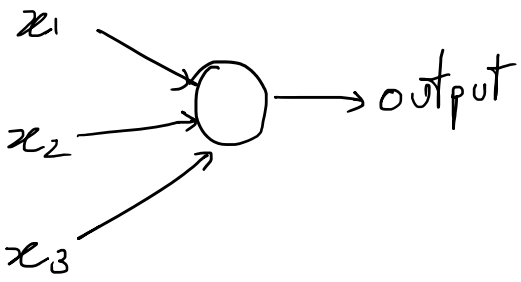
↳ example → sigmoid
→ tanh
→ relu

⇒ The purpose of activation functions is to introduce non-linearities into the network

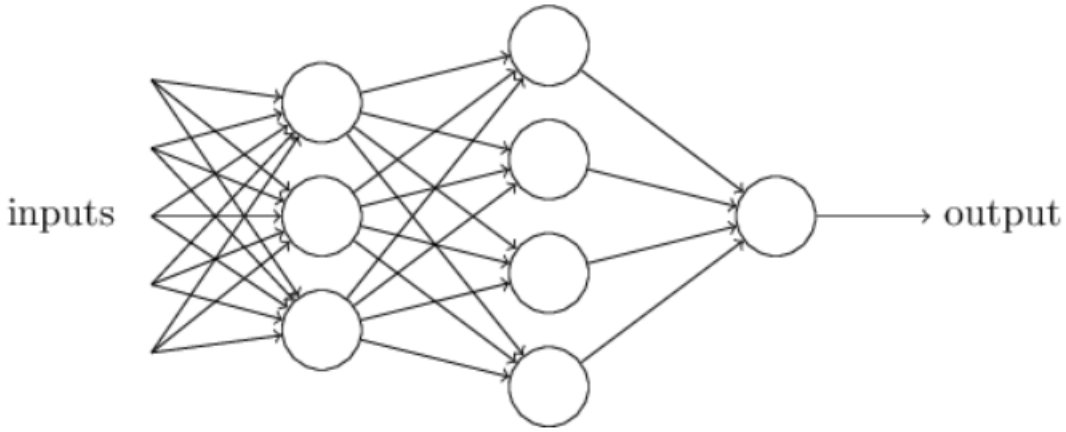
Neural Networks and Deep Learning Handwriting

16 September 2021 10:10

A perceptron takes several binary inputs $x_1, x_2 \ldots$ and produces a single binary output



$$\text{output} = \begin{cases} 0 & \text{if } \sum_{\forall i} w_i x_i < \text{threshold} \\ 1 & \text{if } \sum_{\forall i} w_i x_i \geq \text{threshold} \end{cases}$$



Convolutional Neural Networks

20 September 2021 12:16

Computer Vision

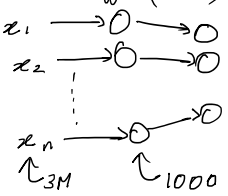
- Self Driving Cars
- Face Recognition
- Style Transfer

1) Image Classification
2) Object Detection
3) Neural Style Transfer

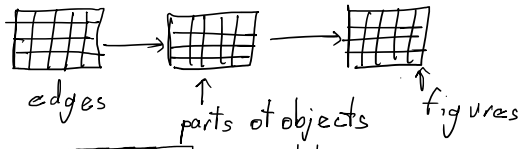
Input Sizes can get really really huge!!

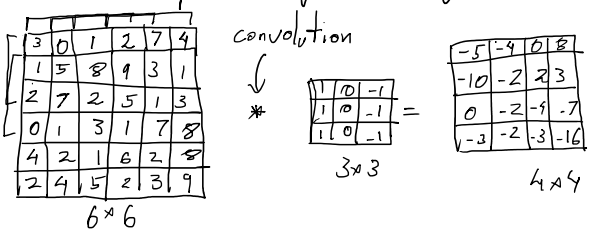
$1000 \times 1000 \times 3 = 3$ million!!

$w^{[1]}$ (1000, 3M) ← 3billion ← what!!



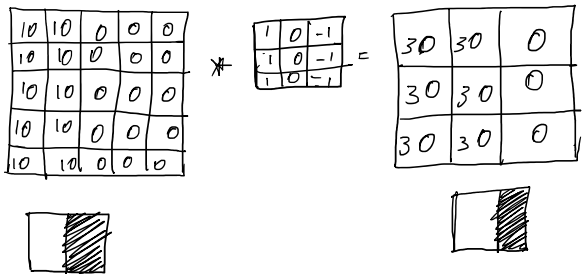
Edge Detection



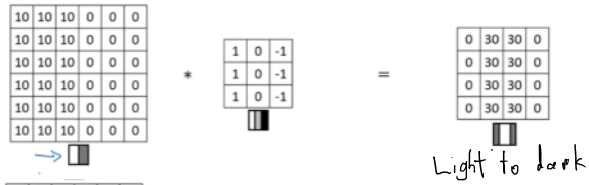


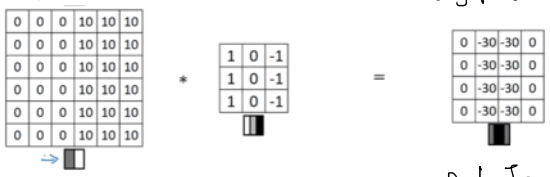
1] python: conv-forward
2] tensorflow: tf.nn.conv2d
3] keras: conv2D

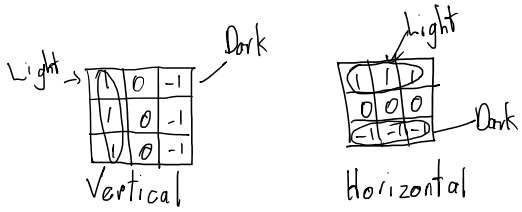
Vertical Edge Detection



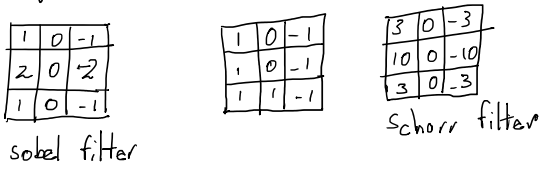
Positive & Negative Edges

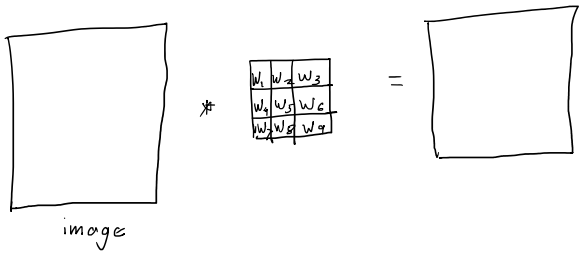




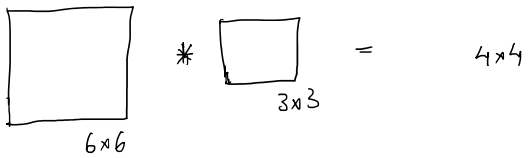


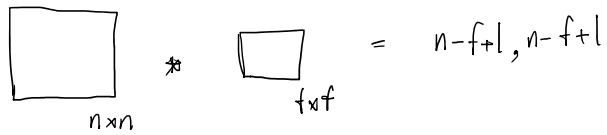
Learning to detect edges





Padding





→ Pixels on corners used less than pixels on centres

→ Shrinks image → can't have more than a few layers

p = padding

↳ O/p image becomes

↳ $n+2p-f+1$, $n+2p-f+1$

Valid & Same convolutions

Valid : $n \times n \;\; * \;\; f \times f \;\; \Rightarrow n-f+1 \times n-f+1$

Same : Pad so that output size is the same as the input size

$n+2p-f+1 \;,\; n+2p-f+1$

$n + 2p - f + 1 = n$

$p = \frac{(f-1)}{2}$

f → usually odd
- → If f is even → asymmetric padding
- Reason → There is a central pixel

Strided Convolutions

$7 \times 7 \;\; * \;\; 3 \times 3 \;\; = 3 \times 3$

stride = 2

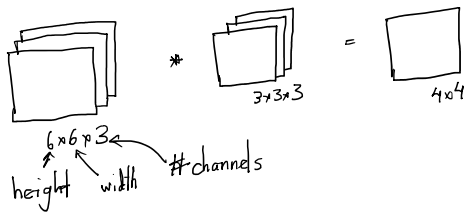
In general if stride is given as 's'

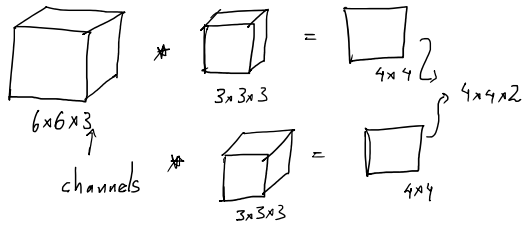
$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

$\lfloor 7 \rfloor$ = floor(7)

Cross Correlation vs Convolution

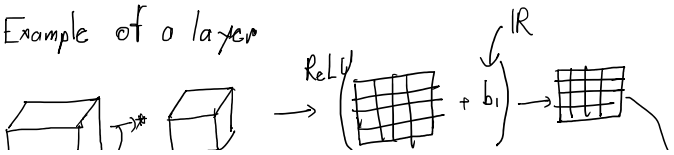
Convolutions on RGB images

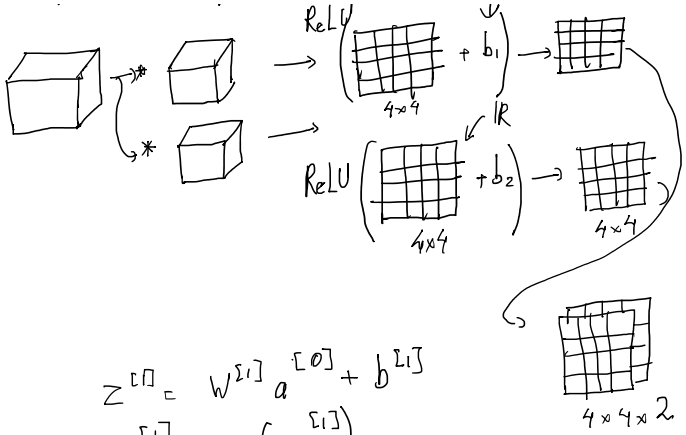




Summary : $n \times n \times n_c \;\; * \;\; f \times f \times n_c \rightarrow n-f+1 \times n-f+1 \times n_c'$ ← #filters

Example of a layer





$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

If layer l is a convolutional layer

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

Input : $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

Output : $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \; ; \; n_W^{[l]} = \left[\frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right]$$

$n_c^{[l]} \rightarrow$ Number of filters

Each filter is : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

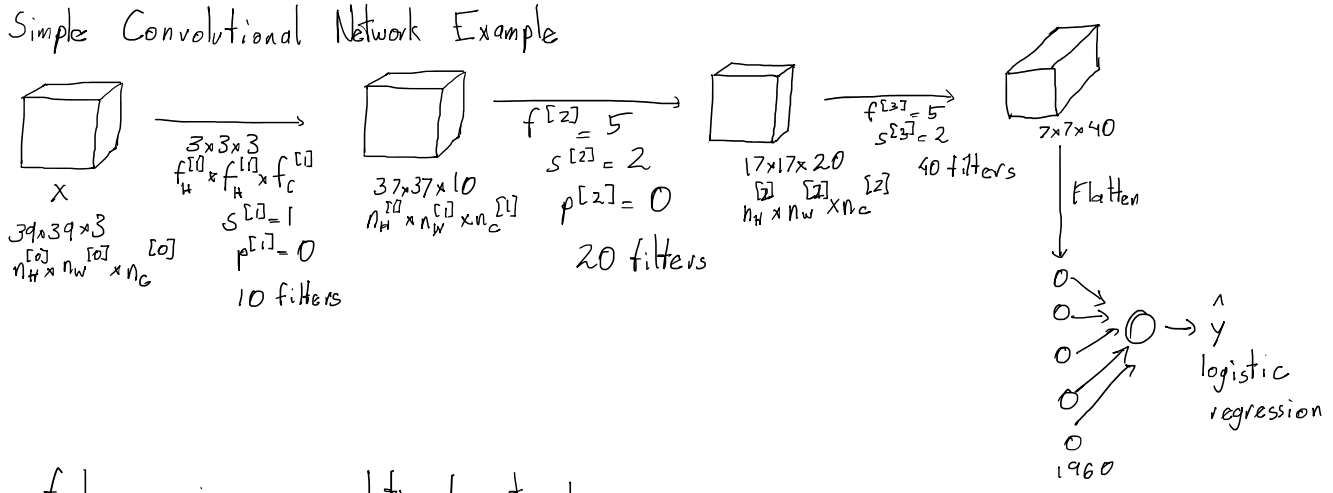
Activations : $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Weights : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$ ← # filters

bias : $n_c^{[l]} - (1, 1, 1, n_c^{[l]})$

Simple Convolutional Network Example



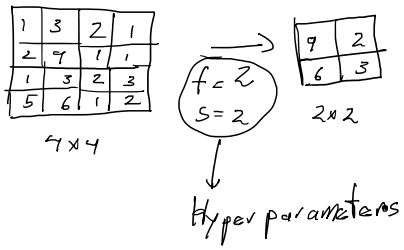
Types of layers in a convolutional network

→ Convolution (Conv)

→ Pooling (POOL)

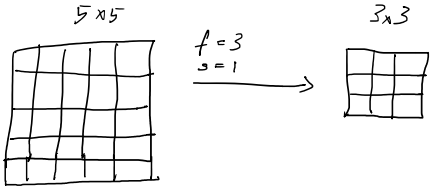
↳ Fully connected (FC)

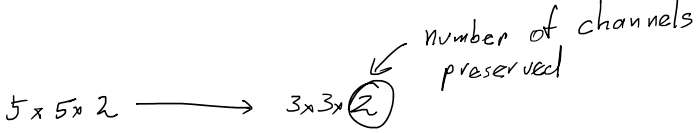
Pooling layer : Max Pooling



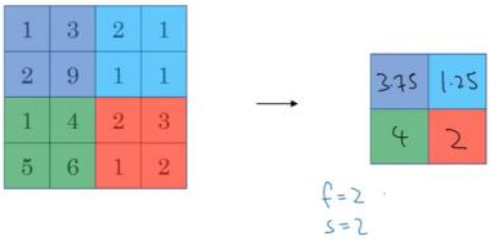
Properties

↳ Hyperparameters but no parameters to learn





Pooling layer : Average Pooling



Max Pooling is used more

Summary of Pooling

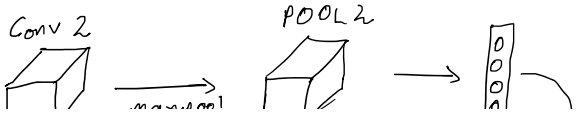
f ⇒ filter size f=2, s=2
s ⇒ stride f=3, s=2

Max or average pooling

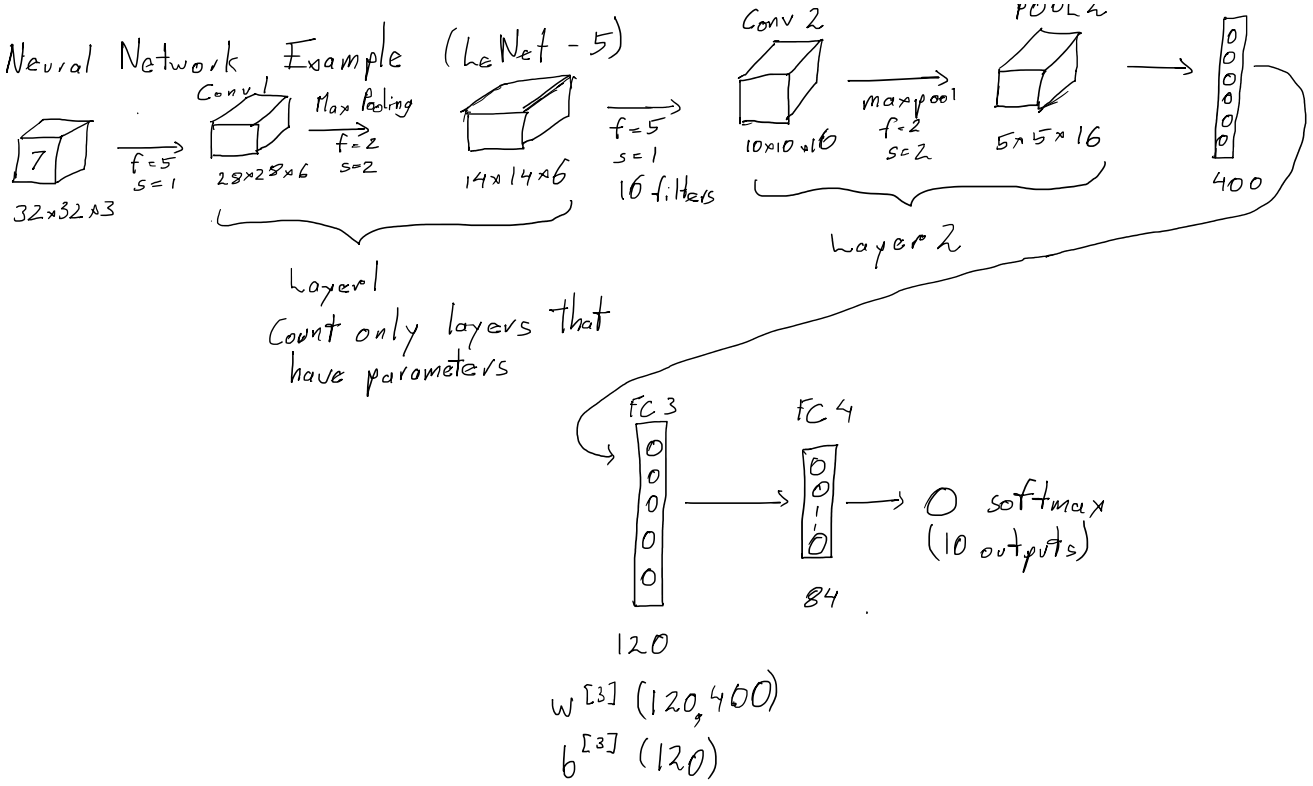
$n_H \times n_w \times n_c$

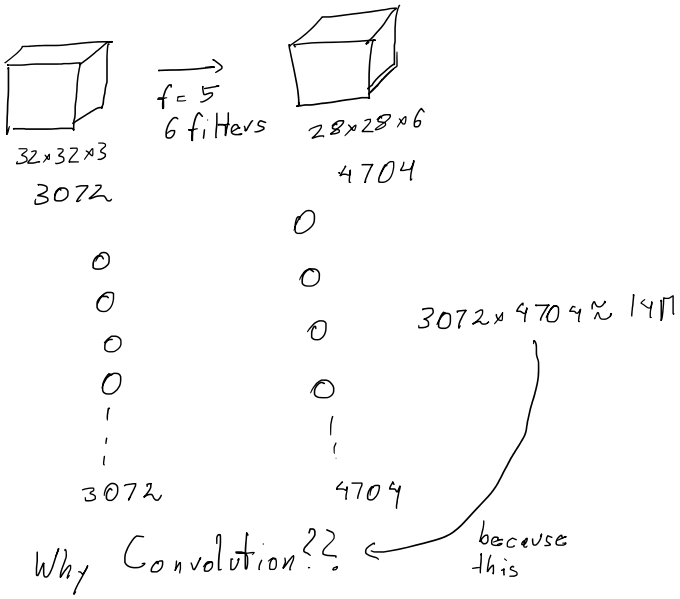
$$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_w - f}{s} + 1 \right\rfloor \times n_c$$

Neural Network Example (LeNet - 5)



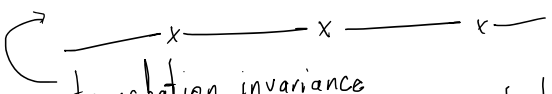
Neural Network Example (LeNet-5)





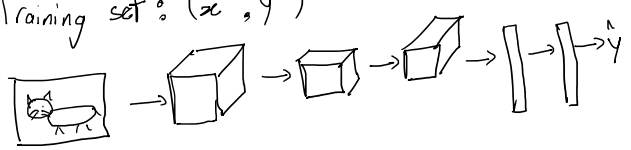
Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image

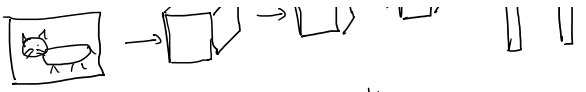
Sparsity of connection: In each layer, each output value depends only on a small number of inputs



translation invariance
↳ NN made more robust

Training set: $(x^{(1)}, y^{(1)})$





$$Cost = \frac{1}{m} \sum L(\hat{y}^{(i)}, y^{(i)})$$

Use gradient descent to optimize parameters to reduce J

Code :-

```
def zero_pad(X, pad):
    X_pad = np.pad(X, (0,0),(pad,pad),(pad,pad),(0,0), mode = 'constant', constant_values = (0,0))
    return X_pad

# Single step of convolution
def conv_single_step(a_slice_prev, W, b):
    s = W * a_slice_prev
    Z = np.sum(s)
    Z = Z + float(b)
    return Z

def conv_forward(A_prev, W, b, hparameters):
    (m, n_h_prev, n_w_prev, n_c_prev) = A_prev.shape
    (f, f, n_c_prev, n_c) = W.shape
    stride = hparameter["stride"]
    pad = hparameters["pad"]
    n_h = int(int(n_h_prev + 2* pad - f)/stride + 1)
    n_w = int(int(n_w_prev + 2* pad - f)/stride + 1)
    Z = np.zeros([m, n_h, n_w, n_c])

    A_prev_pad = zero_pad(A_prev, pad)
    for i in range(m):
        a_prev_pad = A_prev_pad[i]
        for h in range(n_h):
            vert_start = stride * h
            vert_end = vert_start + f
            for w in range(n_w):
                horiz_start = stride * w
                horiz_end = horiz_start + f
                for c in range(n_c):
                    a_slice_prev = A_prev_pad[i, vert_start: vert_end,
                    horiz_start:horiz_end, :]
                    weights = W[:,:,:,c]
                    biases = b[:,:,:,c]
                    Z[i,h,w,c] = conv_single_step(a_slice_prev, weights, biases)

    assert(Z.shape = (m, n_h, n_w, n_c))
    cache = (A_prev, W, b, hparameters)
    return Z, cache

def pool_forward(A_prev, hparameters, mode = "max"):
    (m, n_h_prev, n_w_prev, n_c_prev) = A_prev.shape
    stride = hparameter["stride"]
    n_h = int(int(n_h_prev - f)/stride + 1)
    n_w = int(int(n_w_prev - f)/stride + 1)
    n_c = n_c_prev
    Z = np.zeros([m, n_h, n_w, n_c])
    for i in range(m):
        for h in range(n_h):
            vert_start = stride * h
            vert_end = vert_start + f
            for w in range(n_w):
                horiz_start = stride * w
                horiz_end = horiz_start + f
                for c in range(n_c):
                    a_prev_start = A_prev[i]
                    if mode == "max":
                        A[i, w, h, c] = np.max(a_prev_slice[vert_start: vert_end, horiz_start:
                        horiz_end, c])
                    elif mode == "average":
                        A[i, h, w, c] = np.mean(a_prev_slice[vert_start: vert_end,
                        horiz_start: horiz_end, c])
    cache = (A_prev, hparameters)
    assert(A.shape == (m, n_h, h_w, n_c))
    return A, cache
```

Convolutional Layer Backward Pass

$$dA += \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} W_c \times dZ_{hw}$$

$$dW_c += \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} a_{slice} \times dZ_{hw}$$

$$db = \sum_h \sum_w dZ_{hw}$$

```
def conv_backward(dZ, cache):
    (A_prev, W, b, hparameters) = cache
    (m, n_h_prev, n_w_prev, n_c_prev) = A_prev.shape
    (f, f, n_c_prev, n_c) = W.shape
    stride = hparameters['stride']
    pad = hparameters['pad']
    (m, n_h, n_w, n_c) = dZ.shape
    dA_prev = np.zeros(m, n_h_prev, n_w_prev, n_c_prev)
    dW = np.zeros(f, f, n_c_prev, n_c)
    db = np.zeros(1,1,1, n_c)

    A_prev_pad = zero_pad(A_prev, pad)
    dA_prev_pad = zero_pad(dA_prev, pad)
```

```
    for i in range(m):
        a_prev_pad = A_prev_pad[i]
        da_prev_pad = dA_prev_pad[i]
        for h in range(n_h):
            for w in range(n_w):
                for c in range(n_c):
                    vert_start = stride * h
                    vert_end = vert_start + f
                    horiz_start = stride * h
                    horiz_end = horiz_start + stride
                    a_slice = a_prev_pad[vert_start: vert_end, horiz_start: horiz_end, c]
                    da_prev_pad[vert_start: vert_end, horiz_start: horiz_end, :] += W[:,:,:, c] *
                    dZ[i, h, w, c]
                    dW[:,:,:,c] += a_slice * dZ[i, h,w,c]
                    db[:,:,:,c] +=dZ[i,h,w,c]


        dA_prev[i, :,:,:] = da_prev_pad[pad:-pad, pad:-pad, :]

    assert(dA_prev.shape == (m, n_h_prev, n_w_prev, n_c_prev))
    return dA_prev, dW, db

def create_mask_from_window(x):
    mask = (x == np.max(x))

    return mask

def distribute_value(dz, shape):
    (n_h, n_w) = shape
    average = dz / (n_h * n_w)
    a = np.ones(shape) * average
    return a

def pool_backward(dA,cache,mode="max"):
    (A_prev,hparameters)=cache
    stride=hparameters['stride']
    f=hparameters['f']
    m,n_H_prev,n_W_prev,n_C_prev=A_prev.shape
    m,n_H,n_W,n_C=dA.shape
    dA_prev=np.zeros_like(A_prev)
    for i in range(m):
        a_prev = A_prev[i, :,:,:]
        for h in range(n_h):
            for w in range(n_w):
                for c in range(n_c):
                    vert_start = stride * h
                    vert_end = vert_start + f
                    horiz_start = stride * h
                    horiz_end = horiz_start + stride

                    if mode == "max":
                        a_prev_slice = a_prev[vert_start: vert_end, horiz_start: horiz_end,
                        c]
                        mask = create_mask_from_window(a_prev_slice)
                        dA_prev[i, vert_start: vert_end, horiz_start: horiz_end, c] + = dA[i,
                        h, w, c] * mask
                    elif mode == "mean":
                        da = dA[i, h, w, c]
                        shape = (f,f)
                        dA_prev[i, vert_start: vert_end, horiz_start: horiz_end, c] +=
                        distribute_value(da, shape)

    assert(dA_prev.shape == A_prev.shape)
    return dA_prev
```

Case Studies

22 September 2021 08:48

→ Classic Networks :-

1) Le Net - 5

2) Alex Net

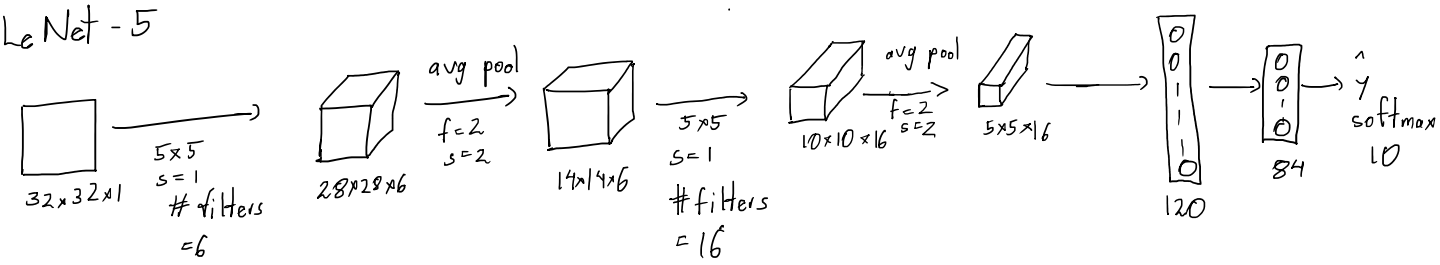
3) VGG

→ ResNet (152 layers)

→ Inception

CLASSIC NETWORKS

1) LeNet - 5



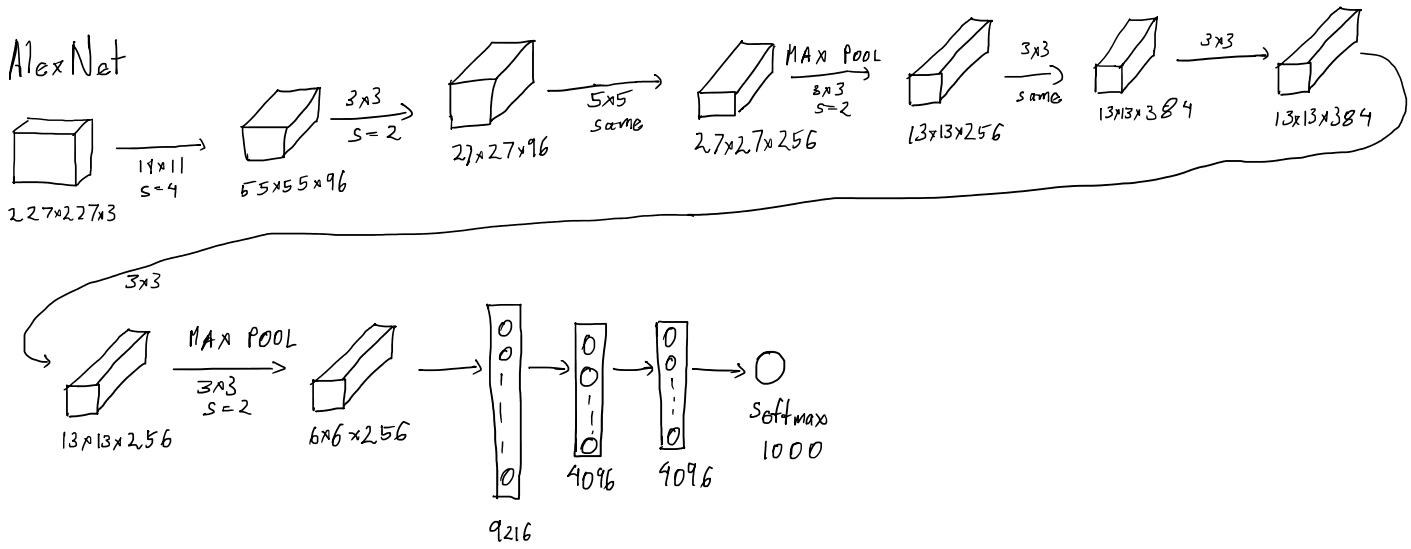
→ ~60K parameters

→ Height & Width ↓ # channels ↑

→ Conv → pool → conv → pool → FC → FC → output

→ Sigmoid / Tanh used & not ReLU

2) AlexNet

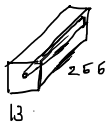


→ Similar to LeNet but much bigger

→ ~60M parameters

→ Multiple GPUs

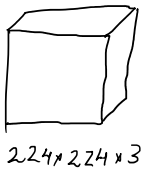
→ Local Response Normalization



3) VGG-16

Conv = 3×3 filter, s = 1, same

Max Pool = 2×2, s = 2



224×224×3 → [CONV 64] ×2 → 224×224×64 → POOL → 112×112×64 → [CONV 128] ×2 → 112×112×128 → POOL → 56×56×128

→ [CONV 256] ×3 → 56×56×256 → POOL → 28×28×256 → [CONV 512] ×3 → 28×28×512 → POOL → 14×14×512 → [CONV 512] ×3 → 14×14×512

→ POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

→ ~138M parameters

→ Simplicity & Uniformity made it very appealing

ResNets

Residual Block

$a^{[l]} \rightarrow a^{[l+1]} \rightarrow a^{[l+2]}$

$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$

$a^{[l+1]} = g^{[l+1]}(z^{[l+1]})$

$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$

$a^{[l+2]} = g^{[l+2]}(z^{[l+2]})$

$a^{[l]} \rightarrow$ Linear $\rightarrow$ ReLU $\xrightarrow{a^{[l+1]}}$ Linear $\rightarrow$ ReLU $\rightarrow a^{[l+2]}$

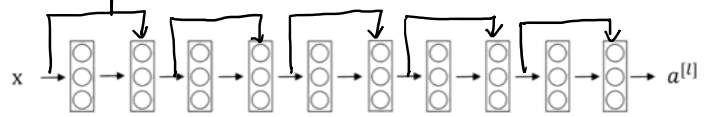
In ResNets however "shortcut"/skip connection

$a^{[l]} \rightarrow$ Linear $\rightarrow$ ReLU $\xrightarrow{a^{[l+1]}}$ Linear $\rightarrow$ ReLU $\rightarrow a^{[l+2]}$

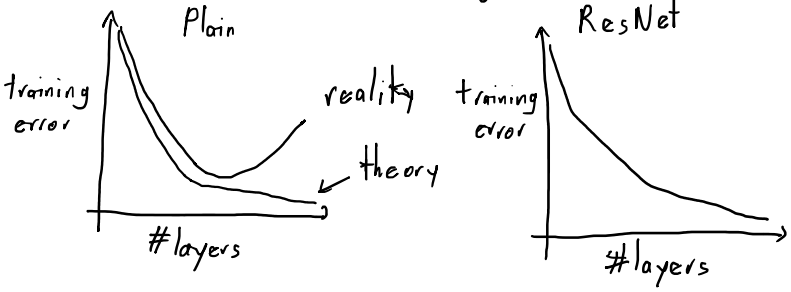
$a^{[l]}$ skips ahead to the second Linear

$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$

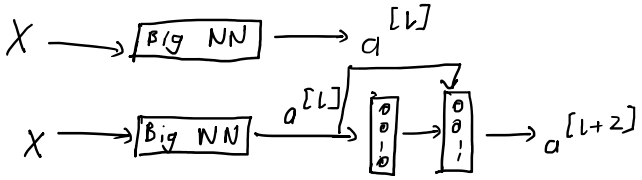
→ Residual Blocks allow you to train much deeper networks



→ 5 residual blocks stacked together



Why ResNets work?



ReLU $\Rightarrow a \geq 0$

$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$ — equal dimensions

generally if not then $g(z^{[l+2]} + W_s a^{[l]})$ — Makes dimension equal

$= g(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$

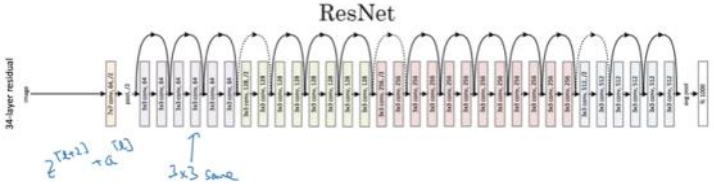
↓ Due to regularisation if $W^{[l+2]} = 0$; $b^{[l+2]} = 0$

$g(a^{[l]}) = a^{[l]}$ (if ReLU)

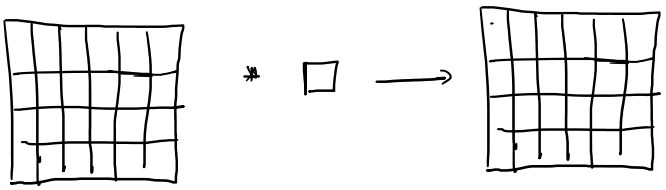
So $a^{[l+2]} = a^{[l]}$ → Therefore "skip" connection

→ So a ResNet doesn't hurt performance atleast

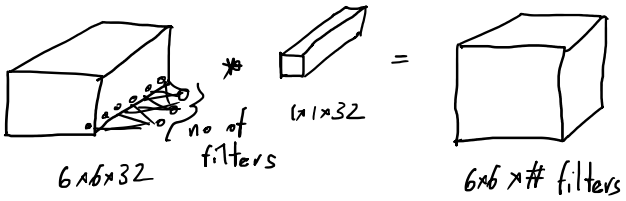
→ Gradient Descent only improves things from there!!



Networks in Networks & 1x1 Convolutions



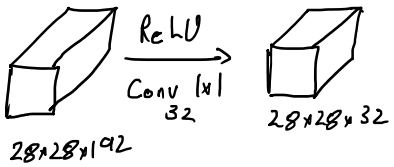
However :-



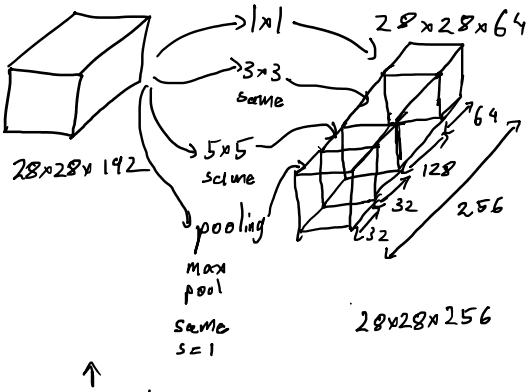
One interpretion :-

→ FC NN that applies to each of the different pixels in image & outputs ∴ → Therefore called network in network !!

Usecase



Inception Network Motivation

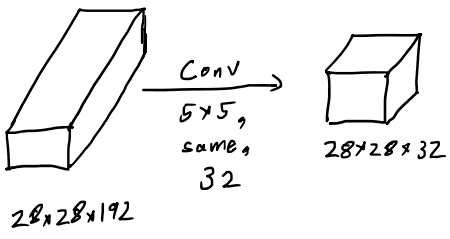


↑

Problem !!

↳ Computational Cost

eg

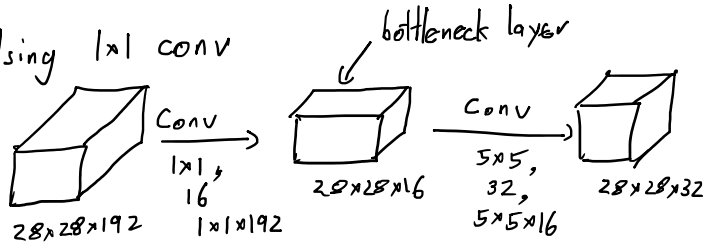


32 filters

filters are 5×5×192

Number of multiplies ⇒ $28 \times 28 \times 32 \times 5 \times 5 \times 192 = 120M$

Using 1×1 conv



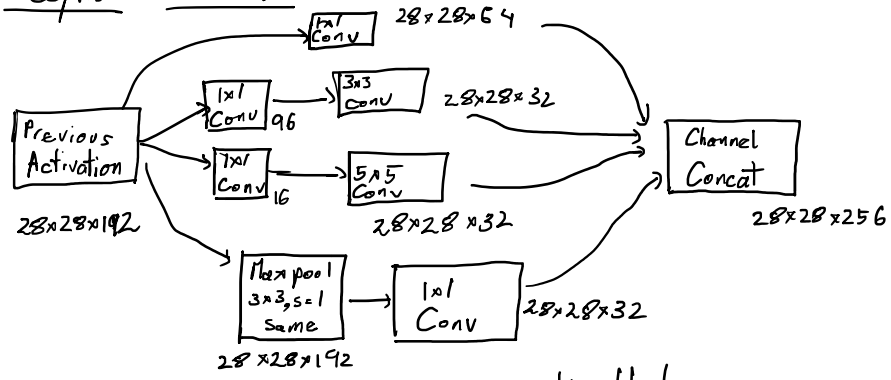
Cost = $(1 \times 1 \times 192 \times 28 \times 28 \times 16) + (5 \times 5 \times 16 \times 28 \times 28 \times 32)$

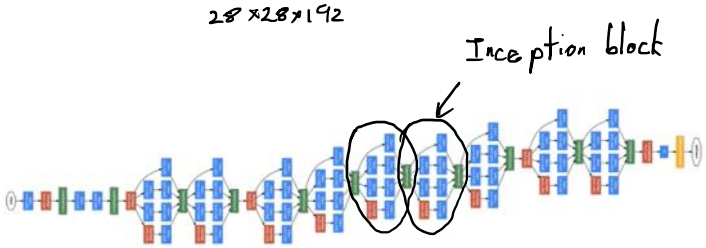
$\sim 2.4M$ $\approx 10M$

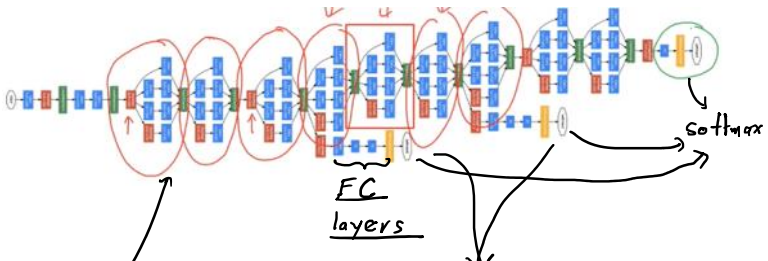
$\approx$ 12.4M multiplications

Reduced!!

Inception Network







Prevents overfitting

googleNet

MobileNet

Motivation of MobileNet

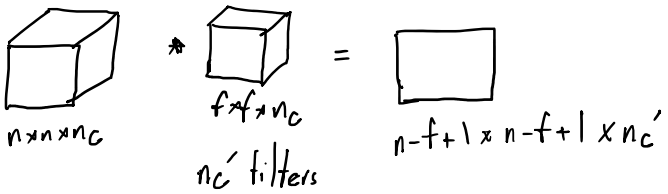
Motivation for MobileNet

→ Low computational cost at deployment

→ Useful for mobile & embedded vision applications

→ Key Idea: Normal vs Depthwise separable convolutions

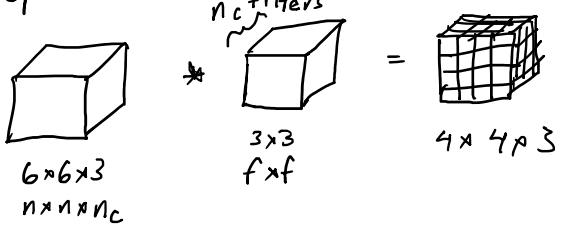
Normal Convolution



Computational Costs = #filter params × # filter positions × # filters

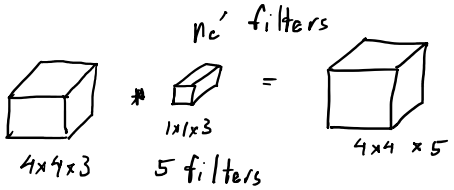
$= f \times f \times n_c \times (n-f+1) \times (n-f+1) \times n_c'$

Depthwise Convolution



Computational cost = $3 \times 3 \times 4 \times 4 \times 3$

$= 432$



Computational Cost = $1 \times 1 \times 3 \times 4 \times 4 \times 5$

$= 240$

Cost of normal convolution : 2160

Cost of depthwise convolution : 432 + 240

672

Typically $= \frac{1}{n_c'} + \frac{1}{f^2}$

$= \frac{1}{5} + \frac{1}{9}$

$= \frac{1}{\quad} + \frac{1}{\quad}$

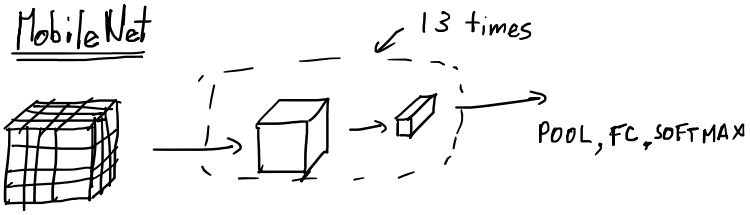
Cost of depthwise convolution : $432 + 240$ / 672

$\frac{672}{2160} = 0.31$

Typically $= \frac{1}{512} + \frac{1}{32}$

$\approx$ 10 times cheaper

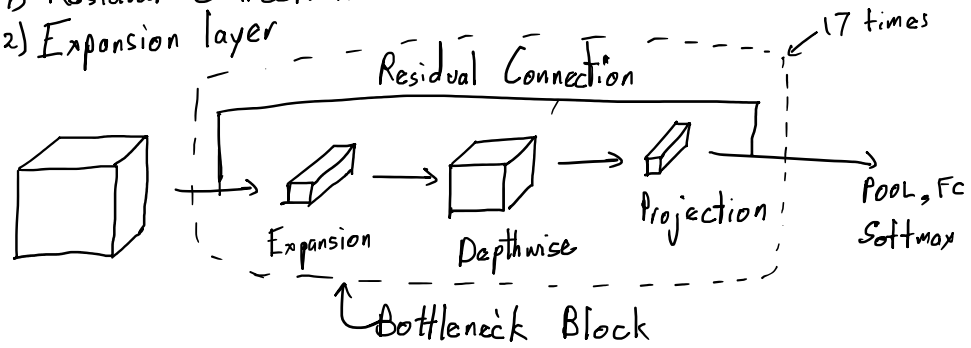
MobileNet

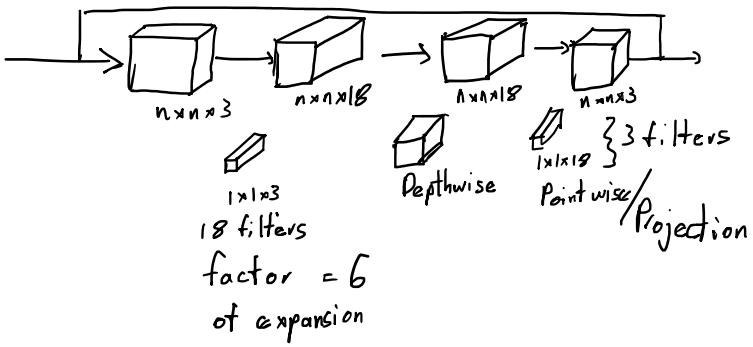


MobileNet v2

Two changes :-

1) Residual Connection
2) Expansion layer





Why did we need bottleneck blocks

→ Expansion increases size of representation for neural network to learn a richer function

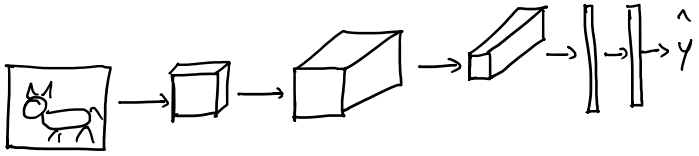
→ Memory Constraint ↓ by projection block by reducing dimensions

Efficient Net

→ Tune as per device

Baseline

Baseline



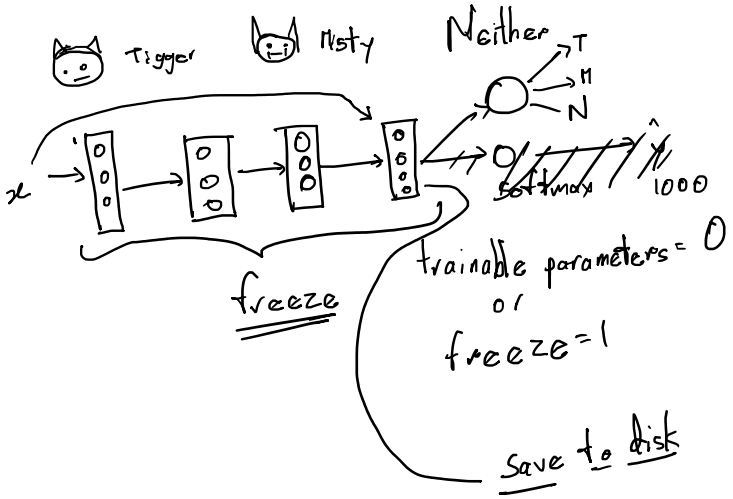
- r → resolution → Use high resolution image
- d → depth → Make NN deeper
- w → width → Make layers wider

▶ Compound scaling (res + depth + width)

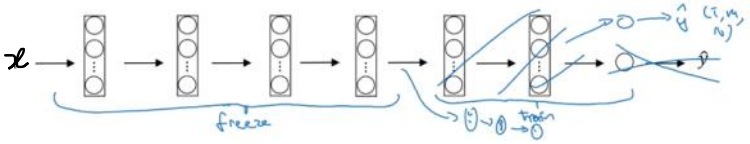
Using Open Source Implementations

↳ Check github!!

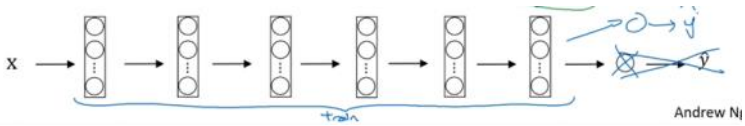
Transfer Learning



If you have more images then



If you have a lot of data then:



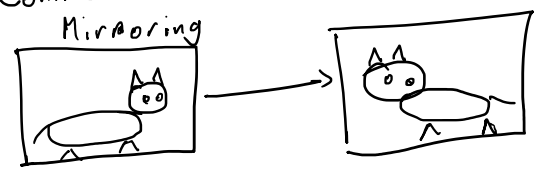
↳ Use downloaded weights only as initializations

Data augmentation

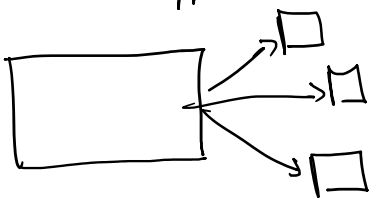
Common methods

Common methods

Mirroring



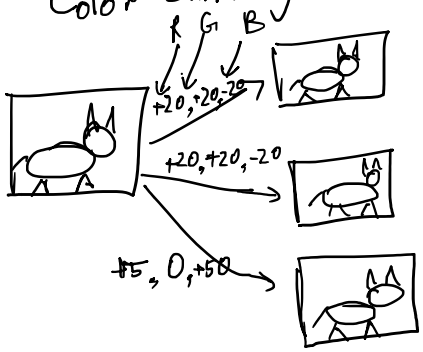
Random Cropping



Used less

- Rotation
- Shearing
- Local Warping
- ⋮

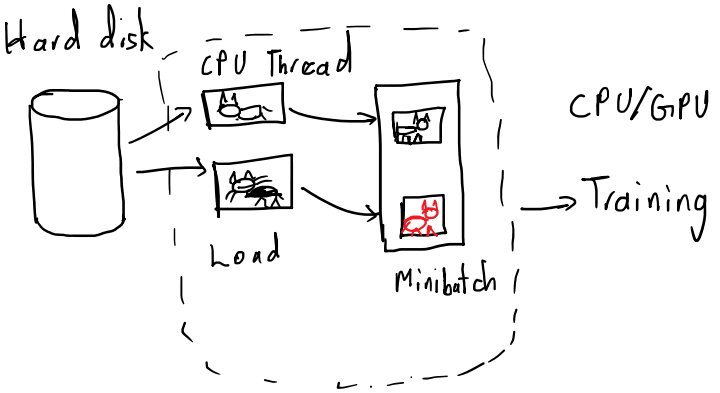
Color Shifting



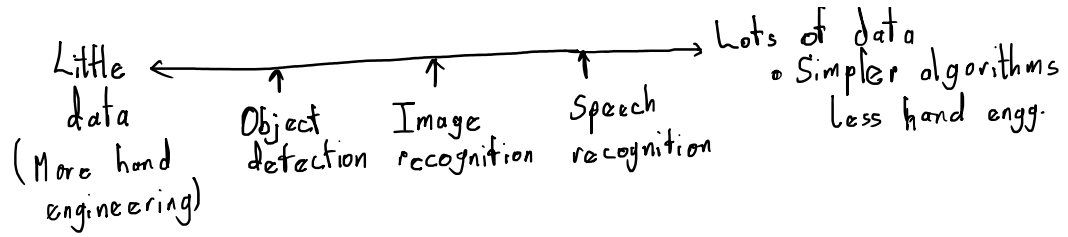
Advanced

PCA → Principal Component Analysis

PCA color augmentation



State of Computer Vision



2 sources of knowledge

→ Labelled data

→ Hand engineered features/network architectures/other components

Tips for doing well on benchmarks

Ensembling

- Train several networks independently & average their outputs
- Never used in production

Multi-crop at test time

- Run classifier on multiple versions of test images & average results

Use open source code

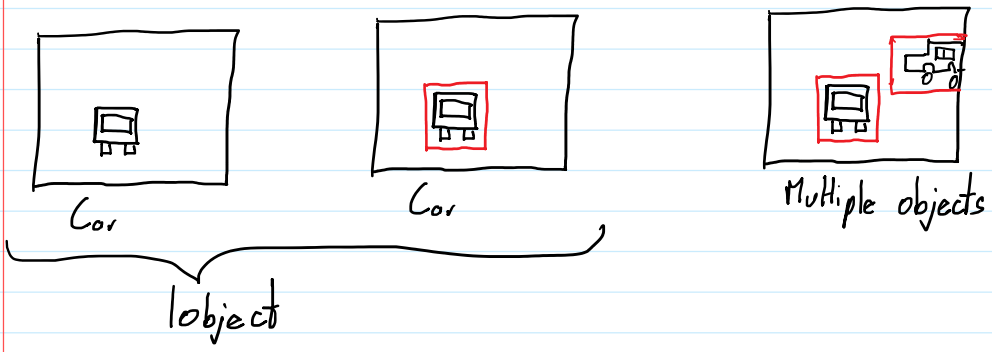
- Use architectures of networks published in the literature
- Use open source implementations if possible
- Use pretrained models and fine-tune on your dataset

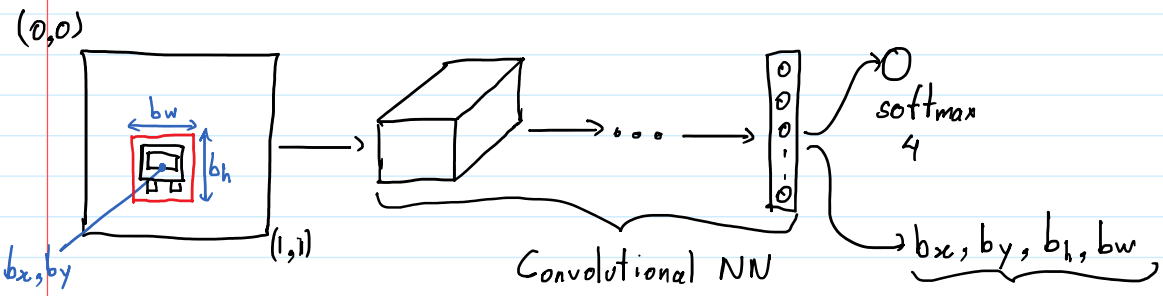
Object Detection

22 September 2021 15:19

Object Localization

Image Classification | Image Classification with localization | Detection





1 - pedestrian
2 - car
3 - motorcycle
4 - background

$b_x = 0.5$
$b_y = 0.7$
$b_h = 0.3$
$b_w = 0.4$

bounding box of object

Defining target label y

1 - pedestrian
2 - car
3 - motorcycle
4 - background

Need to output b_x, b_y, b_h, b_w, class label (1-4)

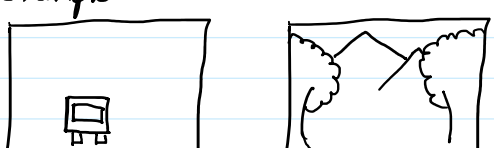
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

p_c → is there an object (1, 2, 3 ?)

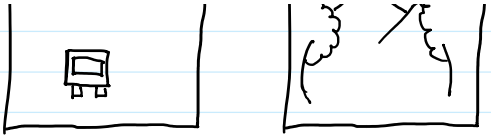
c_1, c_2, c_3 } Assumption — Only 1 object

example

$x =$



$X =$

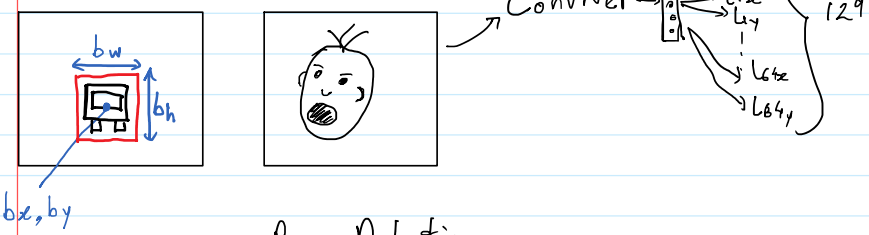


$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$ Don't Care

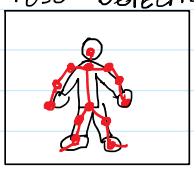
Loss fn

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

Landmark Detection



Pose Detection



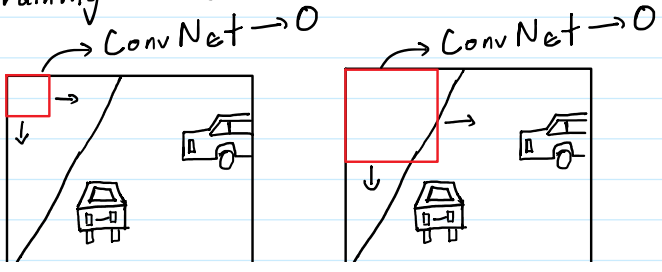
l_{1x}, l_{1y}

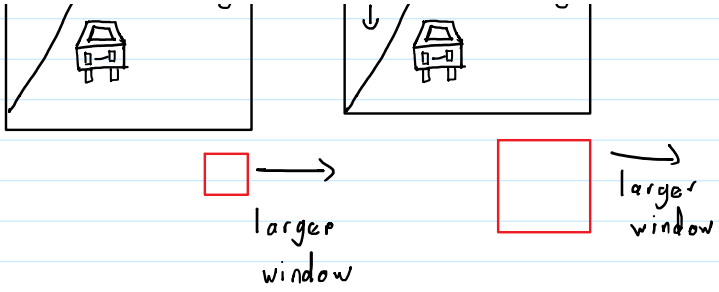
l_{32x}, l_{32y}

Object Detection

Sliding Windows detection algorithm

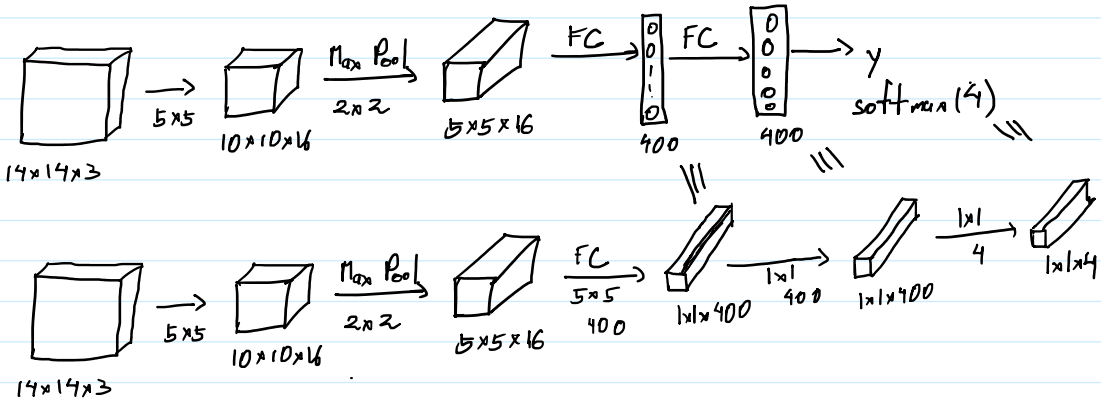
Training Set:

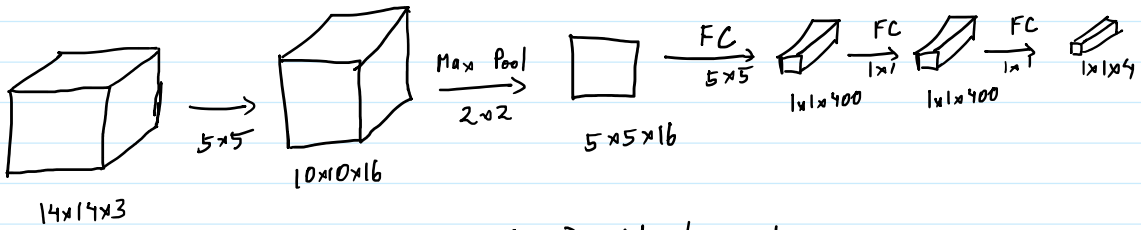




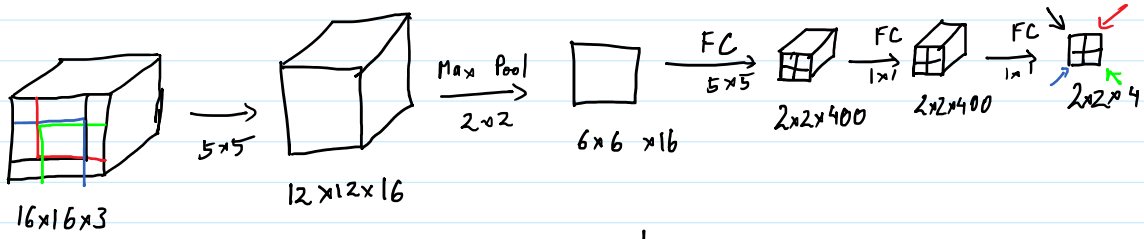
Huge Disadvantage! — Computational Cost is really high

Convolutional Implementation of Sliding Windows





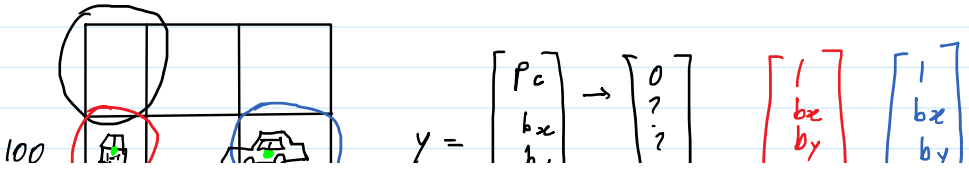
Suppose input image is 16×16×3. We have to run above ConvNet 4 times OR

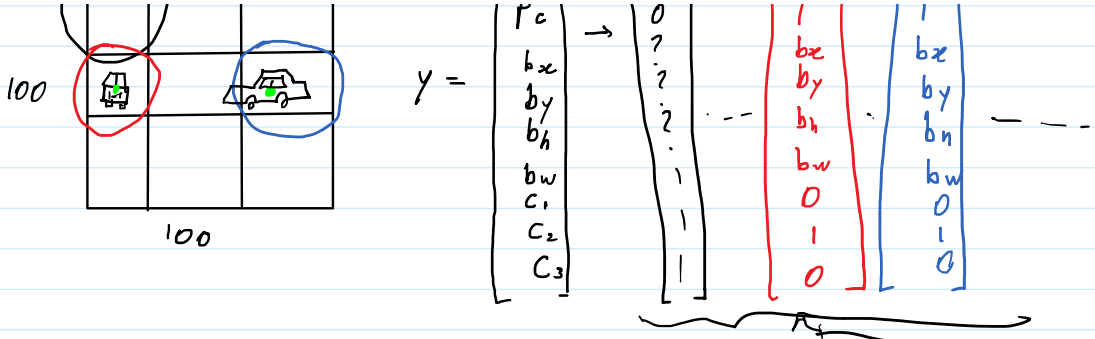


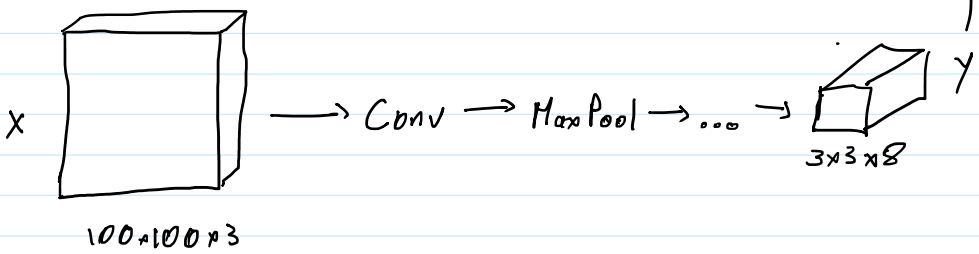
Output Accurate Bounding Box Predictions

Your sliding windows may sometimes capture only part of the object/ miss the object altogether

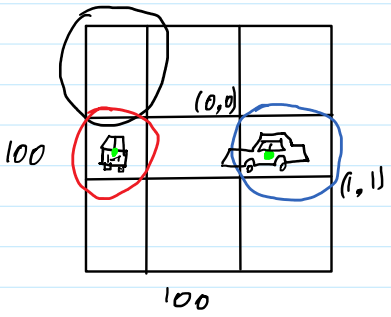
YOLO Algorithm → You Only Look Once







Assign object to grid cell containing its mid point

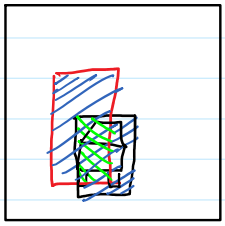


$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

0.4, 0.3 } b/w 0 & 1

0.9, 0.5 } could be > 1

Intersection Over Union



Intersection Over Union (IoU)

$$= \frac{\text{size of } \square}{\text{size of } \square}$$

Correct if IoU ≥ 0.5

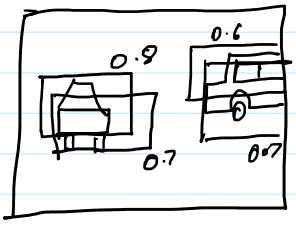
More generally, IoU is a measure of the overlap b/w 2 bounding boxes

Non-max Suppression

Detect 1 object only once

What if every grid cell containing object felt that it had the midpoint?

object felt that it was ...



Non Max Suppression →

Take highest probability
Remove all rectangles with a high IoU with highest probability rectangle

Each o/p prediction is $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$

⇒ Discard all boxes with $p_c \leq 0.6$

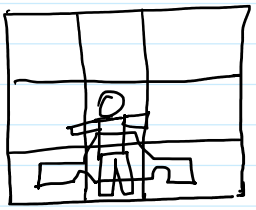
⇒ While there are any remaining boxes

→ Pick box with largest p_c
Output that as a prediction

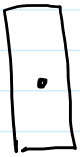
→ Discard any remaining box with IoU ≥ 0.5 with box output in the previous step

Anchor Boxes

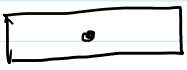
Overlapping objects



Anchor box 1



Anchor box 2



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ \vdots \\ c_3 \end{bmatrix}$$

Anchor box 1

Anchor box 2

Previously

Each object in training image is assigned to grid cell that contains that object's midpoint.

Output y :

3 x 3 x 8

With 2 anchor boxes :-

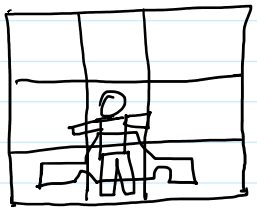
Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

(grid cell, anchor box)

Output y :

3 x 3 x 16 OR

3 x 3 x 2 x 8



$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Car only ??

$$\begin{bmatrix} 0 \\ ? \\ \vdots \\ ? \\ \vdots \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

anchor box 1

anchor level 2

YOLO Algorithm

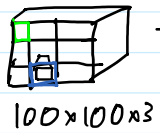
Training

1 - pedestrian

2 - car

3 - motorcycle

$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \end{bmatrix}$ $3 \times 3 \times 2 \times 8$



100 x 100 x 3 → ConvNet → 3 x 3 x 16 → $y =$ $3 \times 3 \times 2 \times 8$

$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \end{bmatrix}$ $\begin{bmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \end{bmatrix}$ $\begin{bmatrix} 0 \\ \vdots \\ 1 \\ b_x \\ b_y \\ b_h \end{bmatrix}$

$$\begin{bmatrix} c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Outputting non max suppressed outputs

→ For each grid cell, get 2 predicted bounding boxes

→ Get rid of low probability predictions

→ For each class (pedestrian, car, motorcycle) use non max suppression to generate the final predictions

Region Proposals

R-CNN

Instead of running sliding windows on all regions, run only on a few chosen ones

Depending on output of segmentation algorithm (~2000)

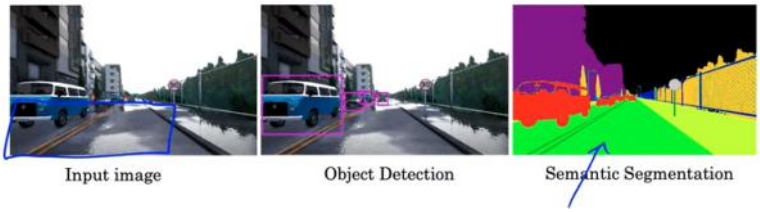
R-CNNs — Propose regions
→ Classify proposed regions one at a time
→ Output label + bounding box

Fast R-CNN — Propose regions
→ Use convolutional implementation of sliding windows to classify all proposed regions

Faster R-CNN — Use convolutional network to propose regions

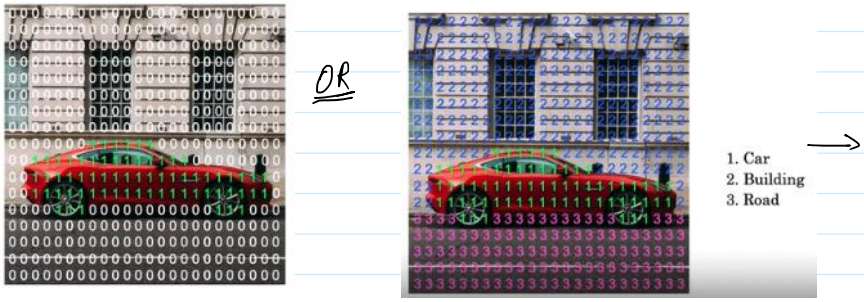
Semantic Segmentation with U-Net

Object Detection vs Semantic Segmentation

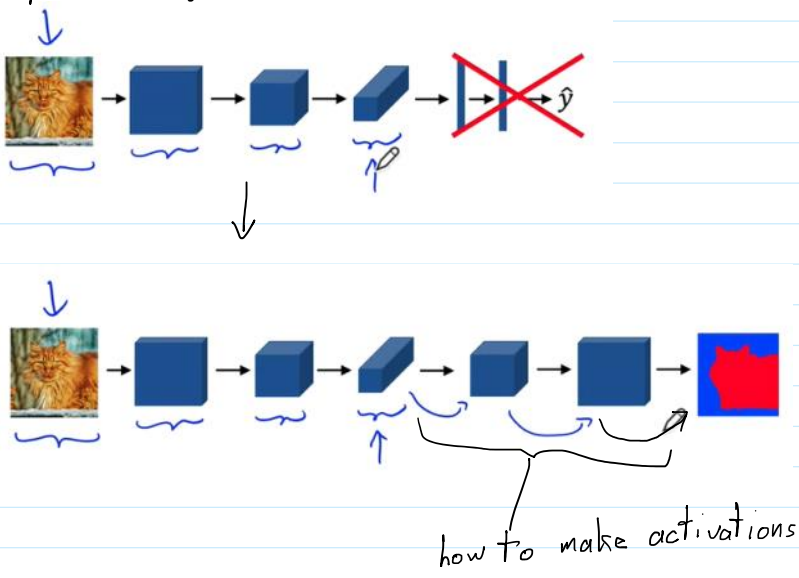


→ Used by self driving cars
→ Segmentation in Chest Radiographs
→ Brain Tumor Detection & Segmentation

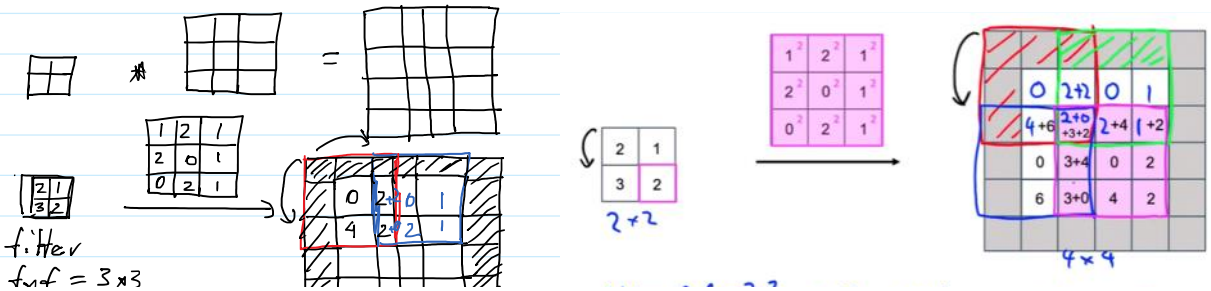
Per pixel class Labels

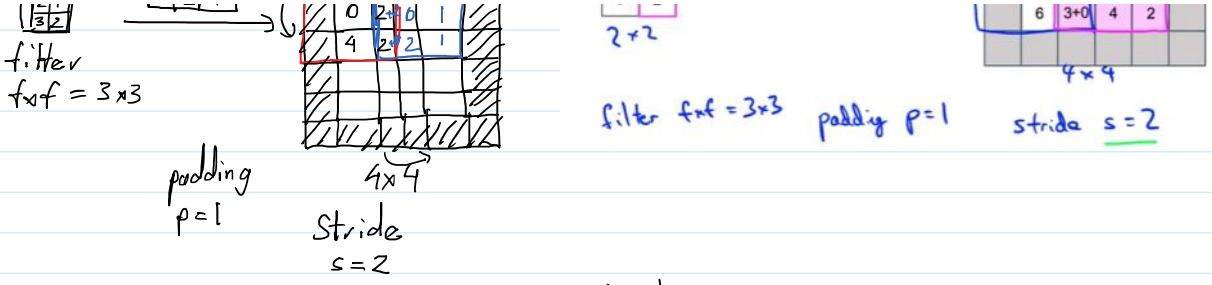


Deep Learning For Semantic Segmentation

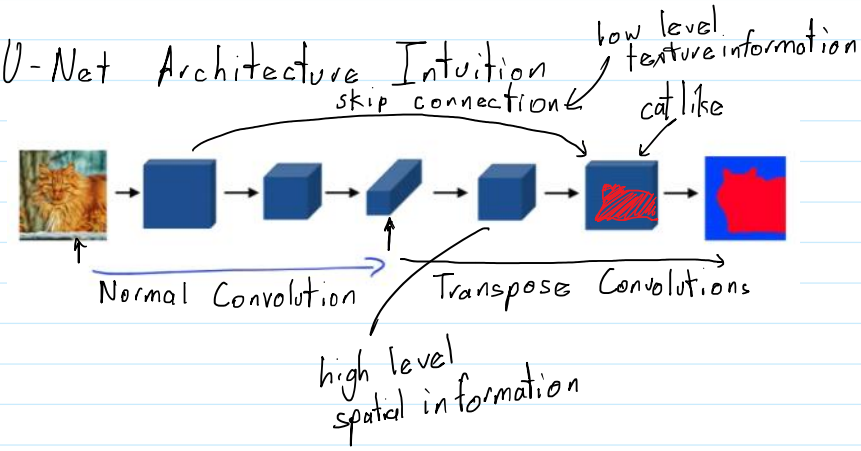


Transpose Convolution

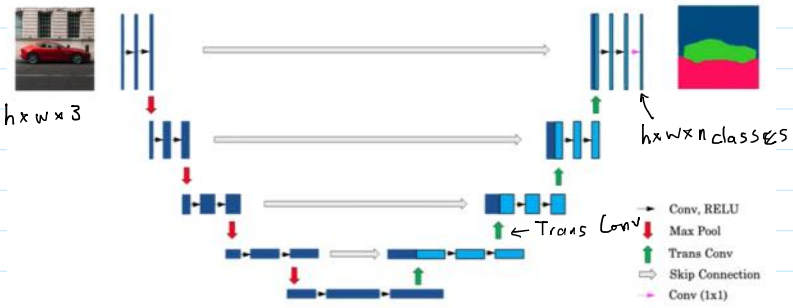


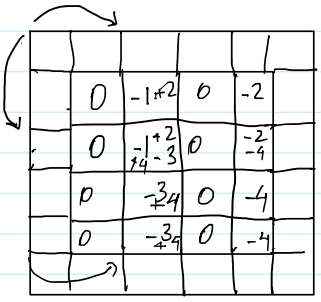


U-Net Architecture Intuition



U-Net Architecture





$-1+2+4-3$

Face Recognition

24 September 2021 08:47

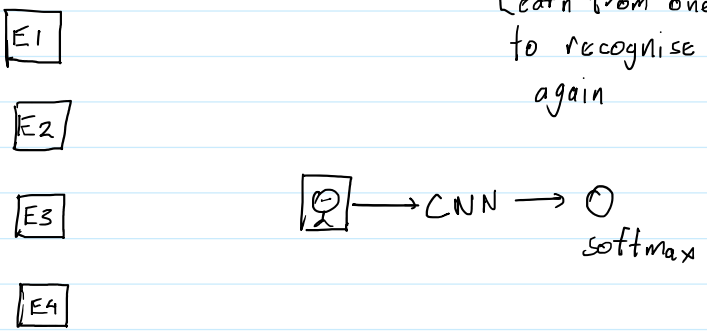
Face Verification vs Face Recognition

Verification
- Input image, name/ID
- Output whether input image is that of the claimed person

Recognition
- Has a database of K persons
- Get an input image
- Output ID if image is any of the K persons (or "not recognised")

One Shot Learning

Learn from one example to recognise the person again



Learning a "similarity" function

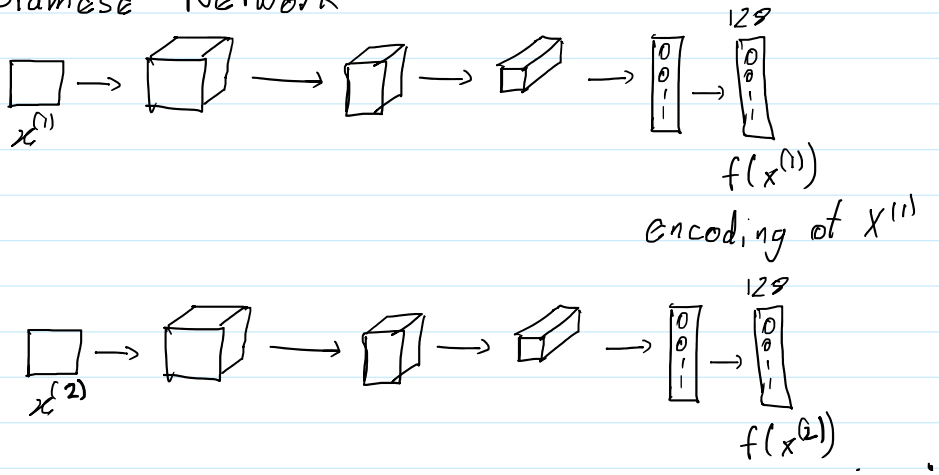
d(img1, img2) = degree of difference b/w images

If $d(img1, img2) \leq \tau$ → 'same'

$\geq \tau$ → 'different'

- Solves 1 shot learning problem
- Solves addition of new person to database

Siamese Network



encoding of $x^{[2]}$

$$d(x^{(1)}, x^{(2)}) = \left\| f(x^{(2)}) - f(x^{(1)}) \right\|_2^2$$

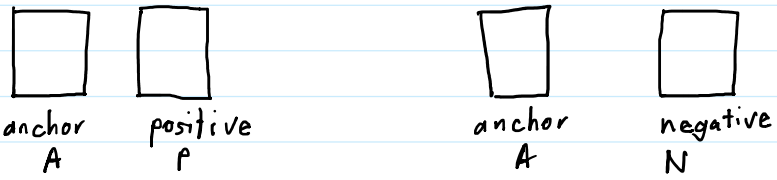
→ Parameters of NN define an encoding $f(x^{(i)})$

→ Learn parameters so that

If $x^{(i)}, x^{(j)}$ are the same person $\left\| f(x^{(i)}) - f(x^{(j)}) \right\|_2^2$ is small

If $x^{(i)}, x^{(j)}$ are different persons, $\left\| f(x^{(i)}) - f(x^{(j)}) \right\|_2^2$ is large

Triplet loss



Want:

$$\frac{\left\| f(A) - f(P) \right\|^2}{d(A,P)} \leq \frac{\left\| f(A) - f(N) \right\|^2}{d(A,N)}$$

$$\left\| f(A) - f(P) \right\|^2 - \left\| f(A) - f(N) \right\|^2 \leq 0 - \alpha$$

α: hparameter (margin)

$f(img) = \vec{k}$: Constant

We don't want this!!

Loss fn

Given 3 images A, P, N

Loss:

$$L(A, P, N) = \max\left(\left\| f(A) - f(P) \right\|^2 - \left\| f(A) - f(N) \right\|^2 + \alpha, 0 \right)$$

$$J = \sum_{i=1}^{m} L(A^{(i)}, P^{(i)}, N^{(i)})$$

Training set: 10k pictures of 1K persons

During training, if A, P, N are chosen randomly,

$d(A, P) + \alpha \leq d(A, N)$ is easily satisfied

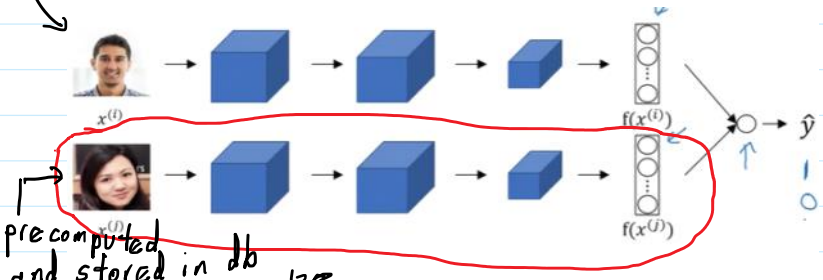
$$\left\| f(A) - f(P) \right\|^2 + \alpha \leq \left\| f(A) - f(N) \right\|^2$$

Choose triplets that are hard to train on

$$d(A,P) + \alpha \leq d(A,N)$$

$$d(A,P) \approx d(A,N)$$

new Another Alternative :-



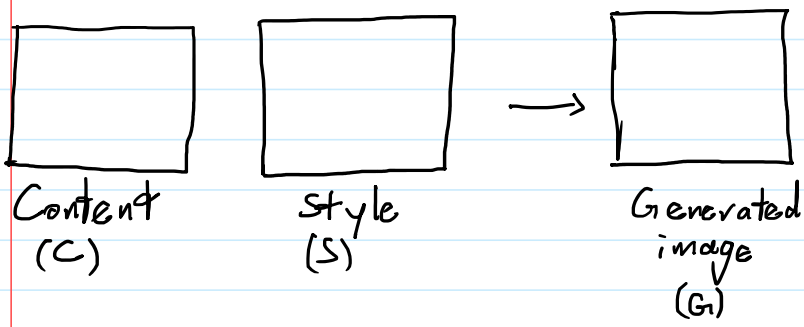
$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_k \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right)$$

$$\frac{\left(f(x^{(i)})_k - f(x^{(j)})\right)^2}{f(x^{(i)})_k - f(x^{(j)})_k}$$

x	y	
	1	"Same"
	0	"Different"
	0	
	1	

Neural Style Transfer

24 September 2021 10:06



Neural Style Transfer Cost Function

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

Find the generated image G

1) Initiate G randomly

G: 100 x 100 x 3

2) Use Gradient Descent to minimize J(G)

$$G = G - \frac{d}{dG}(J(G))$$

Content Cost Function

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

→ Say you chose layer l to compute content cost

→ Use pretroined ConvNet (eg VGG)

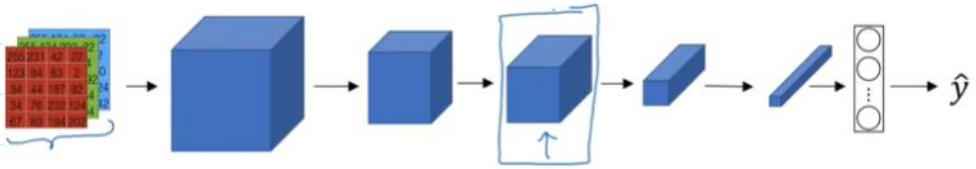
→ Let $a^{[l](C)}$ & $a^{[l](G)}$ be activation of layer l on images

→ If $a^{[l](C)}$ & $a^{[l](G)}$ are similar, both images have similar content

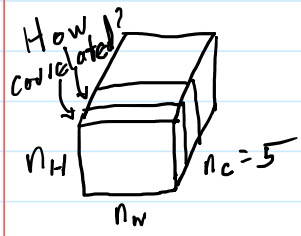
$$J_{content}(C, G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$$

$$J_{content}(C, G) = \frac{1}{2}\underbrace{\| a^{[l](C)} - a^{[l](G)} \|}_{\text{L2 norm}}$$

Style Cost Function



Say you are using layer l's activation to measure "style."
Define style as correlation between activations across channels.



How correlated are the activations across different channels?

Style matrix

Let $a^{[l]}_{i,j,k}$ = activation at (i, j, k). $G^{[l]}$ is $n_C^{[l]} \times n_C^{[l]}$

$G^{[l]}_{kk'}$

$k = 1, 2, \ldots, n_C$

$$G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](S)}_{ijk}\, a^{[l](S)}_{ijk'}$$

gram matrix

$$G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](G)}_{ijk}\, a^{[l](G)}_{ijk'}$$

$$J^{[l]}_{style}(S, G) = \left\| G^{[l](S)} - G^{[l](G)} \right\|_F^2$$

$$= \frac{1}{(2 n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_{k'} \sum_{k} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)$$

$$J_{style}(S, G) = \sum_l \lambda^{[l]} J^{[l]}_{style}(S, G)$$

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

Implementation of CNNs with Tensorflow and Keras

27 September 2021 14:07

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Download & prepare the CIFAR 10 dataset

60000 images in 10 classes (6000 per class) } 50000 training, 10000 test images

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

} Verify the data

Create the Convolutional Base

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape = (32, 32, 3)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation = 'relu'))

model.summary()
```

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(10))

model.summary()
```

} Dense layers

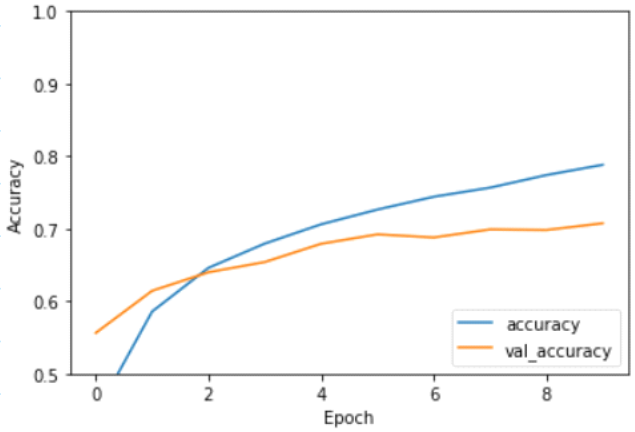
Compile and train the model

```
model.compile(optimizer = 'adam', loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits =
True), metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs = 10, validation_data = (test_images, test_labels))
```

Evaluate the model

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```



Alternatively if the image are to be stored on the disk:-

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

Download & Explore the Dataset

```
import pathlib
dataset_url ="https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin = dataset_url, untar = True)
data_dir = pathlib.Path(data_dir)

image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)

roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[0]))
```

Create a dataset

```
batch_size = 32
img_height = 180
img_width = 180

train_ds = tf.keras.preprocessing.image_dataset_from_directory(data_dir,
                                                               validation_split = 0.2,
                                                               subset = "training",
                                                               seed = 123,
                                                               image_size = (img_height, img_width),
                                                               batch_size = batch_size
                                                               )
```

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(data_dir,
                                                             validation_split = 0.2,
                                                             subset = "validation",
                                                             seed = 123,
                                                             image_size = (img_height, img_width),
                                                             batch_size = batch_size
                                                             )
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

Visualizing the data

```
for image_batch, labels_batch in train_ds:
        print(image_batch.shape)
        print(labels_batch.shape)
```

Configure the dataset for best performance

Dataset.cache()

→ keeps images in memory after they're loaded off disk during the first epoch
→ Ensure dataset does not become a bottleneck while training your model
→ If your dataset is too large to fit into memory, you can use this method to create a performant on-disk cache

Dataset.prefetch()

→ Overlaps data preprocessing & model execution while training

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE
```

Standardize the data

```
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
```

↖ 2 ways to use this layer

↳

```
normalized_ds = train_ds.map(lambda x, y:
(normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixels values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image)
```

```
normalized_ds = train_ds.map(lambda x, y:
(normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixels values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image)
```

Or you can use the layer inside your model definition

```
num_classes = 5

model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255,
input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

layer 1 for normalising inputs

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])


model.summary()


epochs = 10
history = model.fit(train_ds, validation_data = val_ds, epochs)
```

Training

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Visualising results

Data Augmentation :-

```
data_augmentation = keras.Sequential(
```

```
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(img_height,
                                                              img_width,
                                                              3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)

model = Sequential([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Dropout layer introduced

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

epochs = 15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

Check this link for details
https://www.tensorflow.org/tutorials/images/classification

Save and Load Models

27 September 2021 15:47

Setup

```
pip install pyyaml h5py # Required to save models in HDF5 format

import os
import tensorflow as tf
from tensorflow import keras

print(tf.version.VERSION)
```

Get an example dataset

```
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

train_labels = train_labels[:1000]
test_labels = test_labels[:1000]

train_images = train_images[:1000].reshape(-1, 28, 28) / 255
test_images = test_images[:1000].reshape(-1, 28, 28) / 255
```

Define a model

```
# Define a simple sequential model
def create_model():
  model = tf.keras.models.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(784,)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10)
  ])

  model.compile(optimizer='adam',
                loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=[tf.metrics.SparseCategoricalAccuracy()])

  return model

# Create a basic model instance
model = create_model()

# Display the model's architecture
model.summary()
```

Save checkpoints during training

Create a tf.keras.callbacks.ModelCheckpoints callback that saves only during training

```
checkpoint_path = "training_1/cp.cpkt"
checkpoint_dir = os.path.dirname(checkpoint_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath = checkpoint_path, save_weights_only = True, verbose = 1)

model.fit(train_images, train_labels, epochs = 10, validation_data = (test_images, test_labels), callbacks =
[cp_callback])

os.listdir(checkpoint_dir)
```

Now create a fresh untrained model & evaluate it on the test set

```
model = create_model()

loss, acc = model.evaluate(test_images, test_labels, verbose = 2)
```

↳ ~ 10%

Now load the weights from the checkpoint & re-evaluate

```
model.load_weights(checkpoint_path)

loss, acc = model.evaluate(test_images, test_labels, verbose = 2)
```

↳ ~ 87%

Checkpoint Callback Options

```
# Include the epoch in the file name (uses `str.format`)
checkpoint_path = "training_2/cp-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

batch_size = 32

# Create a callback that saves the model's weights every 5 epochs
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    verbose=1,
    save_weights_only=True,
    save_freq=5*batch_size)

# Create a new model instance
model = create_model()

# Save the weights using the `checkpoint_path` format
model.save_weights(checkpoint_path.format(epoch=0))

# Train the model with the new callback
model.fit(train_images,
          train_labels,
          epochs=50,
          batch_size=batch_size,
          callbacks=[cp_callback],
          validation_data=(test_images, test_labels),
          verbose=0)
```

Train a new model
Save uniquely named checkpoints once every 5 epochs

Now look at the resulting checkpoints & choose the latest one

```
os.listdir(checkpoint_dir)
latest = tf.train.latest_checkpoint(checkpoint_dir)
latest

model = create_model()
model.load_weights(latest)

loss, acc = model.evaluate(test_images, test_labels, verbose = 2)
```

What are these files?

What are these files?

- One or more shards that contain your model's weights
- An index file that indicates which weights are stored in which shard

Manually save weights

Manually saving weights with the Model.save_weights method. By default, tf.keras—and save_weights in particular—uses the TensorFlow checkpoint format with a .ckpt extension

```
model.save_weights('./checkpoints/my_checkpoint')
model = create_model()
model.load_weights('./checkpoints/my_checkpoint')
loss, acc = model.evaluate(test_images, test_labels, verbose = 2)
```

Save the entire model

```
# Create and train a new model instance.
model = create_model()
model.fit(train_images, train_labels, epochs=5)

# Save the entire model as a SavedModel.
!mkdir -p saved_model
model.save('saved_model/my_model')
```

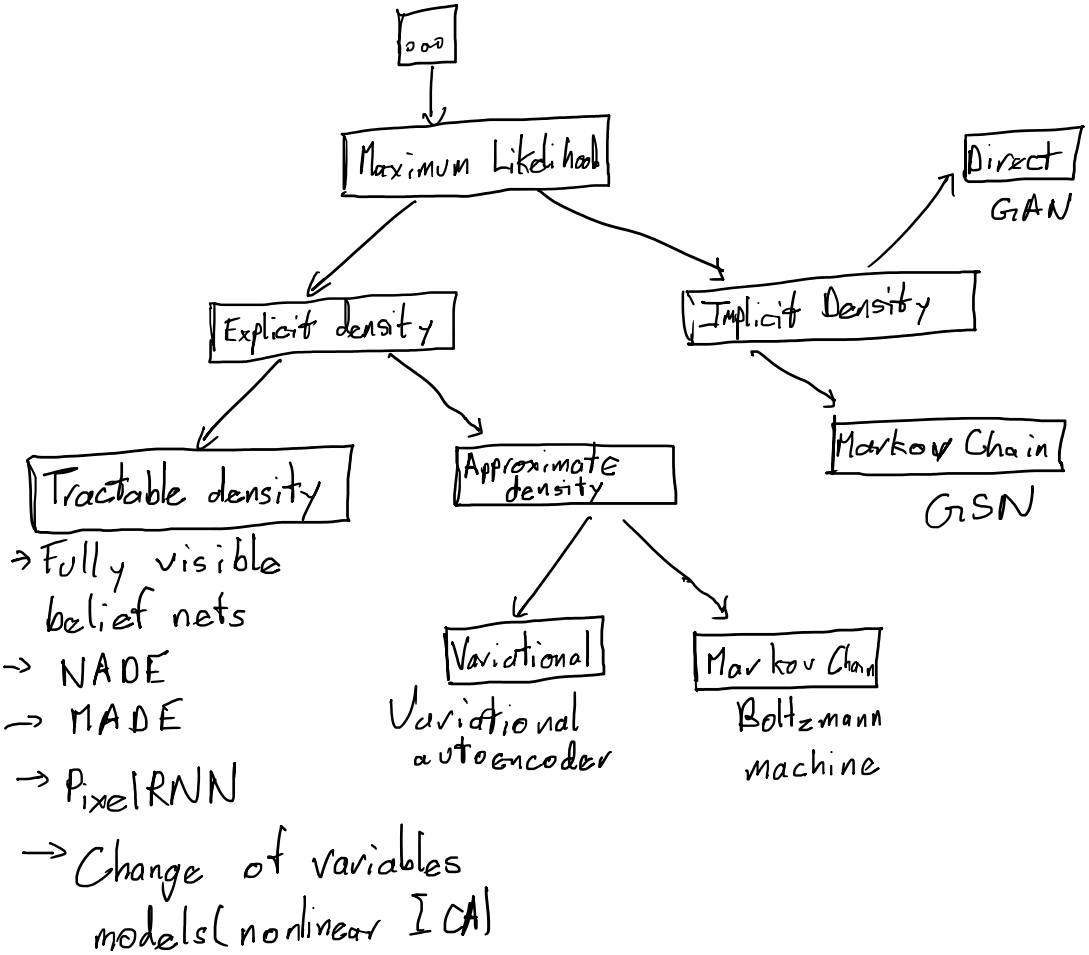
Reload a fresh Keras model from the saved model

```
new_model = tf.keras.models.load_model('saved_model/my_model')

# Check its architecture
new_model.summary()
```

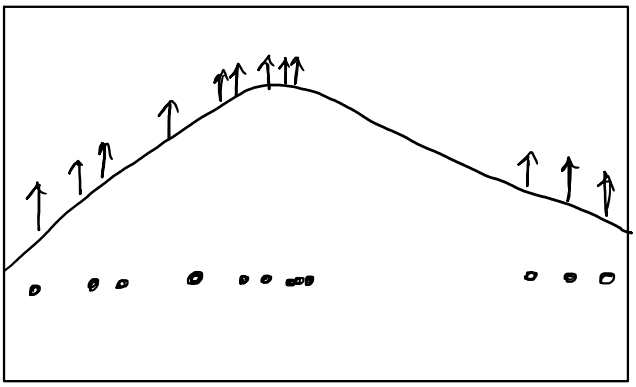
Generative Adversarial Networks

29 September 2021 12:39

Taxonomy of Generative Models



Maximum Likelihood



$$\theta^* = \arg\max_{\theta} E_{x \sim p_{data}} \log \underbrace{p_{model}(x|\theta)}_{\text{Distribution}}$$

input: x

controlled by parameter θ

Fully Visible Belief Nets

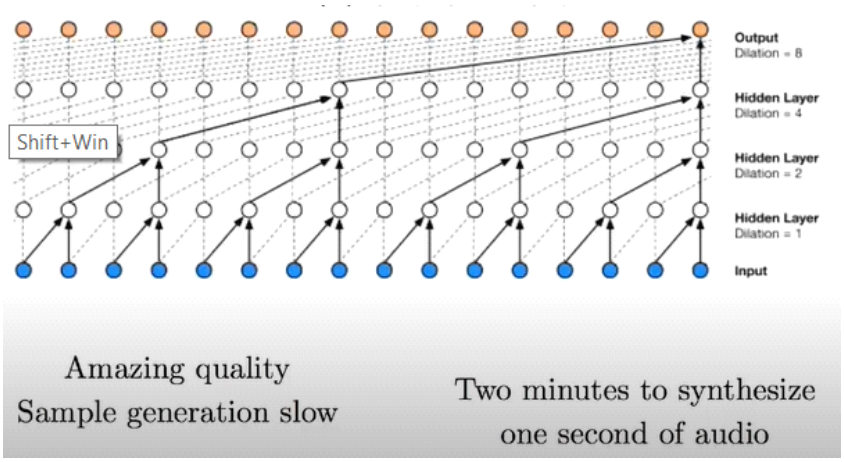
→ Explicit Formula based on chain rule

$$p_{model}(x) = p_{model}(x_1) \prod_{i=2}^{n} p_{model}(x_i | x_1, \ldots x_{i-1})$$

→ Disadvantages

- $O(n)$ sample generation cost
- Generation not controlled by a latent code

Wave Nets



Change of Variables

$$y = g(x) \Rightarrow p_x(x) = p_y(g(x)) \left| \det \left(\frac{\partial g(x)}{\partial x} \right) \right|$$

eg → Non linear ICA

Disadvantages

- Transformation must be invertible
- Latent dimension must match visible dimension

Variational Autoencoder

$$\log p(x) \geq \log p(x) - D_{KL}(q(z) \| p(z|x))$$

$$= E_{z \sim q} \log p(x, z) + H(q)$$

Disadvantages

- Not asymtotically consistent unless q is perfect
- Samples tend to have lower quality

GANs

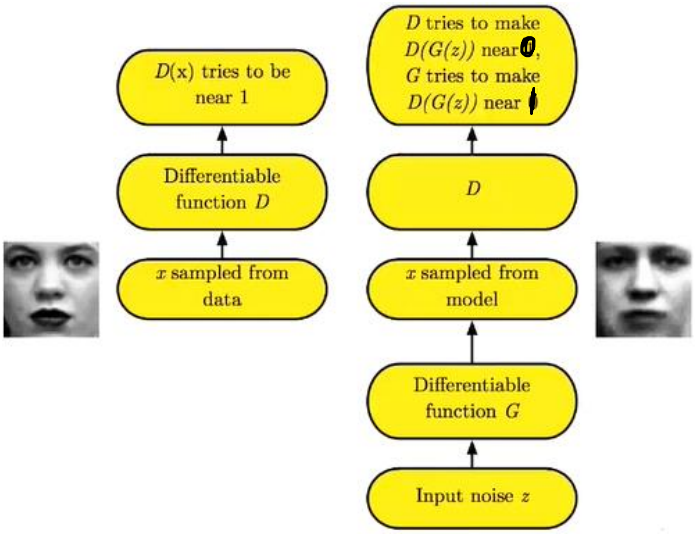
→ Use a latent code — describes everything that follows

→ Asymtotically consistent

→ No Markov Chains needed

→ Often regarded as producing the best samples

How do GANs work?



Adversaries

Generator — primary model we're interested in learning

Discriminator — inspect a sample & understand if its real or fake

Generator Network

$x = G(z; \theta^{(G)})$

z → x

- Must be differentiable
- No invertibility requirement
- Trainable for any size of z
- Some guarantees require z to have higher dimension than x
- Can make x conditionally Gaussian given z but need not do so

Training Procedure

- Use SGD-like algorithm of choice (Adam) on two minibatches simultaneously:
  - A minibatch of training examples
  - A minibatch of generated samples
- Optional: run k steps of one player for every step of the other player.

Minimax Game

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$
$$J^{(G)} = -J^{(D)}$$

-Equilibrium is a saddle point of the discriminator loss
-Resembles Jensen-Shannon divergence
-Generator minimizes the log-probability of the discriminator being correct

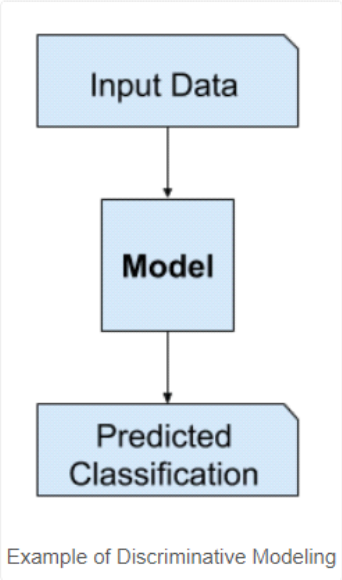
Solution

→ Assume both densities are non-zero everywhere
  → If not, some input values x are never trained so some values of $D(x)$ have undetermined behavior
  → Solve for where the functional derivatives are zero

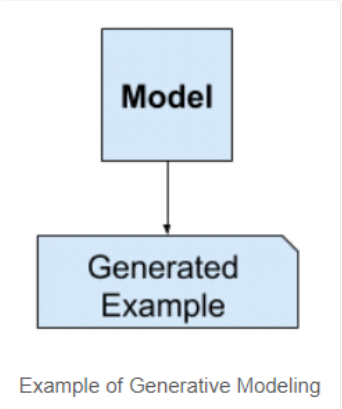
$$\frac{\partial}{\partial D(x)} J^{(D)} = 0$$

Optimal $D(x)$ for any $p_{data}(x)$ & $p_{model}(x)$ is always

$$D(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)}$$



Example of Discriminative Modeling



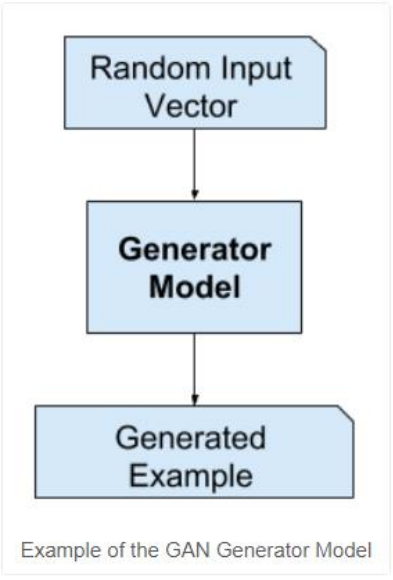
Example of Generative Modeling

For example, a single variable may have a known data distribution, such as a Gaussian distribution, or bell shape. A generative model may be able to sufficiently summarize this data distribution, and then be used to generate new variables that plausibly fit into the distribution of the input variable.

The GAN model architecture involves two sub-models: a *generator model* for generating new examples and a *discriminator model* for classifying whether generated examples are real, from the domain, or fake, generated by the generator model.

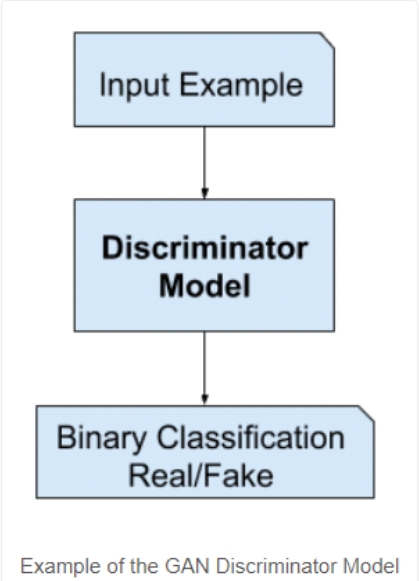
- **Generator**. Model that is used to generate new plausible examples from the problem domain.
- **Discriminator**. Model that is used to classify examples as real (*from the domain*) or fake (*generated*).

The Generator Model

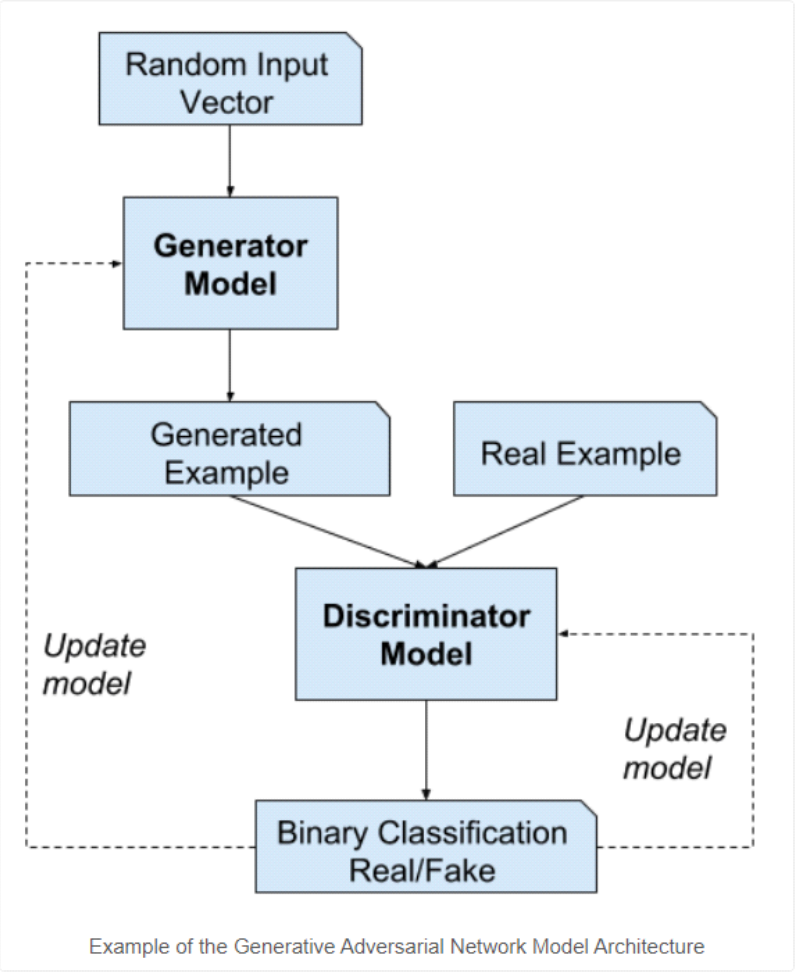


Example of the GAN Generator Model

The Discriminator Model



Example of the GAN Discriminator Model



Example of the Generative Adversarial Network Model Architecture

CSPNet

14 January 2022 15:33

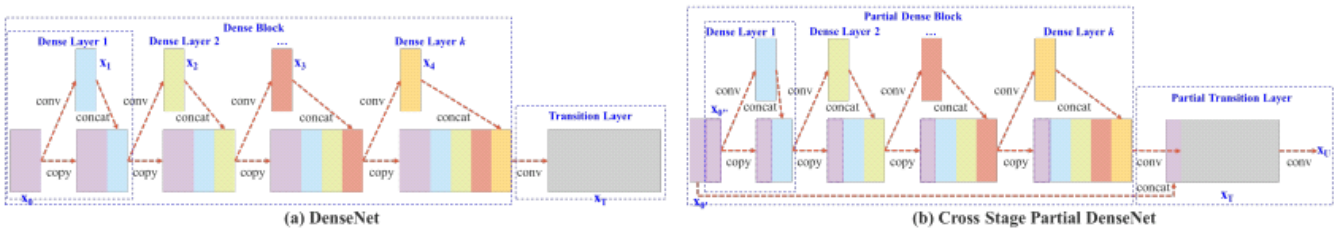
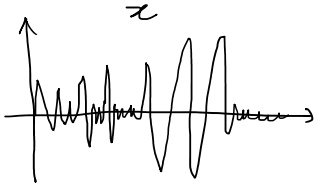


Figure 2: Illustrations of (a) DenseNet and (b) our proposed Cross Stage Partial DenseNet (CSPDenseNet). CSPNet separates feature map of the base layer into two part, one part will go through a dense block and a transition layer; the other one part is then combined with transmitted feature map to the next stage.

Recursive Neural Networks

06 October 2021 10:24

Speech Recognition  → The quick brown fox jumps over the lazy dog

Music generation ∅ → 

DNA Sequence analysis AGTCCCTGTG... → AGTCCCTGTG...

Machine Translation French → English

Video activity recognition  → Running

Named entity recognition Yesterday Harry Potter met Hermione Granger → Yesterday Harry Potter met Hermione Granger

Motivating example

x : Harry Potter and Hermione Granger invented a new spell
$x^{<1>}$ $x^{<2>}$ ... $x^{<t>}$

y : 1 1 0 1 1 0 0 0 0

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ ... $T_x = T_y = 9$

$X^{(i)<t>}$ $T_x^{(i)} = 9$

↓

$y^{(i)<t>}$ $T_y^{(i)} = 9$

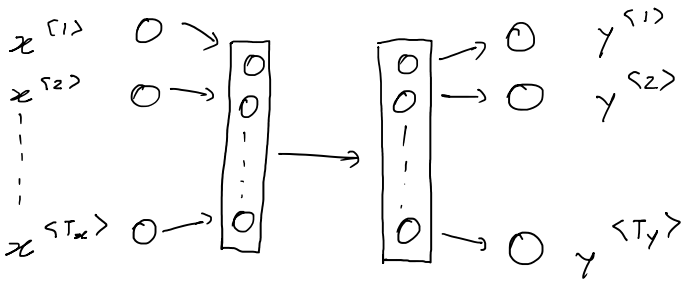
Vocabulary

a	1
aaron	2
⋮	
and	367
⋮	
harry	4075
⋮	
zulu	10000

$X^{<1>}$ → [0, 0, ⋮, 1, ⋮, 0, 0, ⋮] 4075

One hot encoding

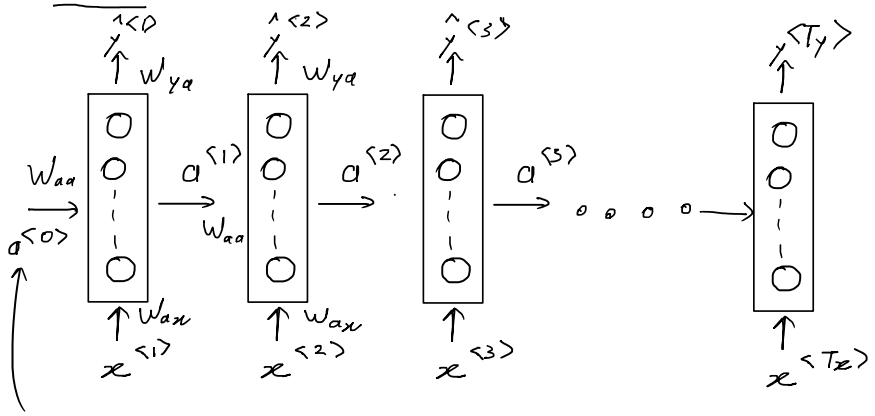
Why not a standard neural network??



Problems:

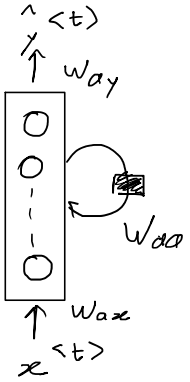
- Inputs, Outputs can be different lengths in different examples
- Doesn't share features learned across different positions of text.

RNN



Vector of zeros

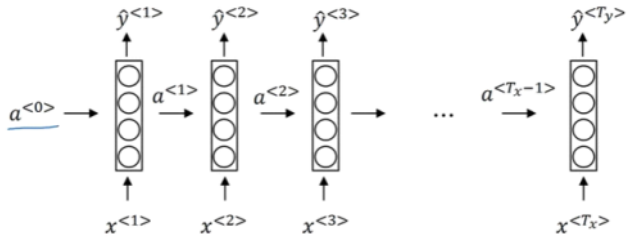
111



He said, "Teddy Roosevelt was great"

He said, "Teddy bears on sale"

Bidirectional RNNs (BRNNs)



$a^{<0>} = \vec{0}$

$$a^{<1>} = g(W_{aa}\, a^{<0>} + W_{ax}\, x^{<1>} + b_a) \quad \leftarrow \tanh / \text{ReLU (more common)}$$

$$\hat{y}^{<1>} = g(W_{ya}\, a^{<1>} + b_y) \quad \leftarrow \text{sigmoid / softmax}$$

$$a^{<t>} = g(W_{aa}\, a^{<t-1>} + W_{ax}\, x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}\, a^{<t>} + b_y)$$

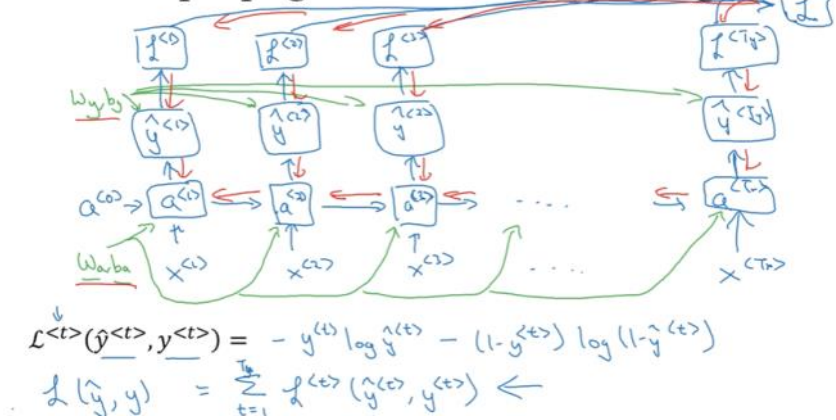
$$a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$$

W_a (100, 10100) $= [W_{aa} \; W_{ax}]$ (100 × 100 | 100 × 10000)

$[a^{<t-1>}, x^{<t>}]$ (100, 10100) $= \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$ (100 × 10000)

Backpropagation Through Time

Forward propagation and backpropagation

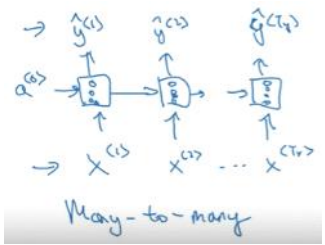


$$\mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} \log \hat{y}^{<t>} - (1 - y^{<t>}) \log (1 - \hat{y}^{<t>})$$

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

Examples of RNN architectures

$T_x = T_y$

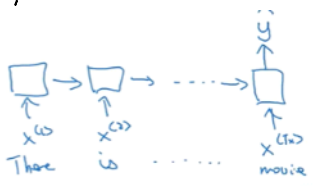


Many-to-many

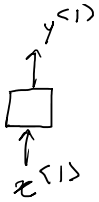
Sentiment Classification

x = text

y = 0/1

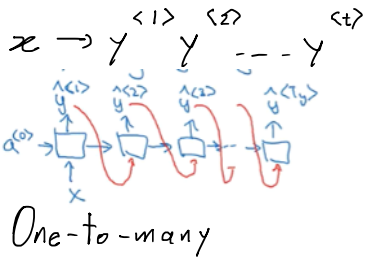


Many-to-one



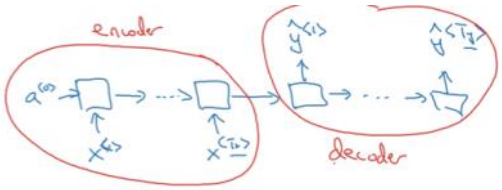
One-to-one
(Standard nn)

Music Generation



One-to-many

Machine Translation



Many to many

Language Model & Sequence Generation

Speech recognition

The apple & pair salad

The apple & pear salad ✓

$P(\text{sentence}) = ?$

$P(y^{<1>}, y^{<2>}, \ldots y^{<t>})$

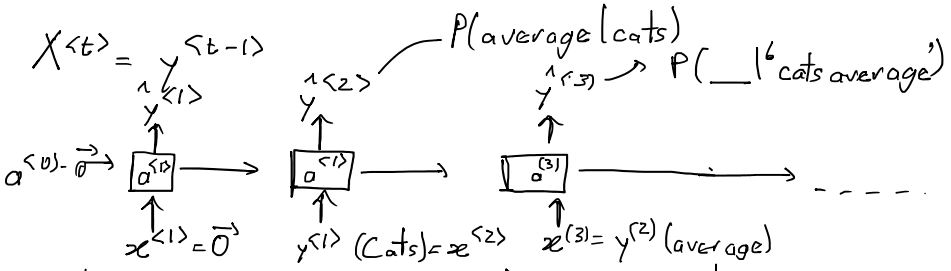
Training set: Large Corpus of english text

Tokenize

Cats average 15 hours of sleep a day <EOS>

$y^{<1>}$ $y^{<2>}$ ... $y^{<n>}$

The Egyption ~~Mau~~ <UNK> is a breed of cat <EOS>



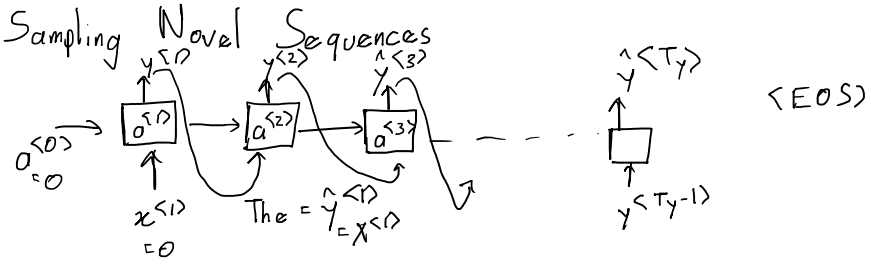
Cats average 15 hours of sleep a day <EOS>

$L(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$

$$\mathcal{L} = \sum_t \mathcal{L}^{\langle t \rangle}(y, y^{\langle t \rangle})$$

$$P(y^{(1)}, y^{(2)}, y^{(3)}) = P(y^{(1)})\, P(y^{(2)} | y^{(1)})\, P(y^{(3)} | y^{(1)}, y^{(2)})$$

Sampling Novel Sequences



P(a) P(aaron) . . . P(zulu) P(<UNK>) = np.random.choice

→ Word level RNN
→ Character level RNN

Vocabulary = [a, b, c, . . . z, _, ,, ., ;, . . . A, . . . Z]

Pros & Cons

Pros ⟶ Able to assign probability to unknown words

Cons ⟶ Much longer sentences

Word level >> Character level

Vanishing Gradients with RNNs

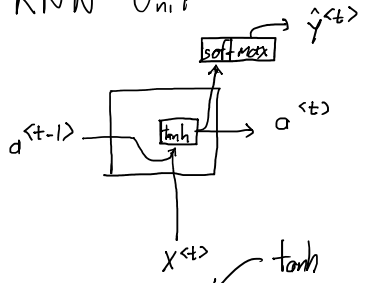
The cat which _ _ _ _ _ , was full

The cats which _ _ _ _ _ , were full

Exploding gradients → NaNs → gradient clipping

Gated Recurrent Unit

RNN Unit.



$$a^{\langle t \rangle} = g(W_a [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_a)$$

(g: tanh)

GRU (simplified)

c ⇒ memory cell

$c^{\langle t \rangle} = a^{\langle t \rangle}$

$\tilde{c}^{\langle t \rangle} = \tanh(W_c [c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c)$

$\Gamma_u = \sigma(W_u [c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$

↳ update

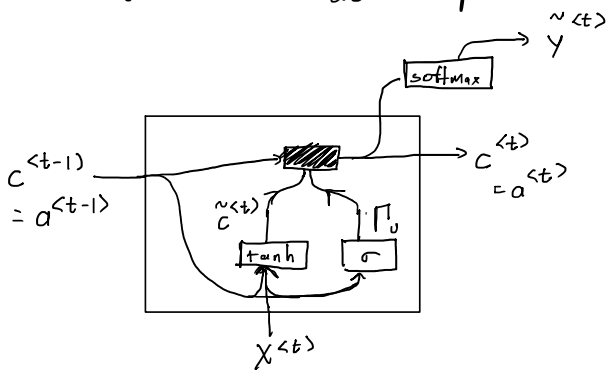
$c^{\langle t \rangle}$ = The cat, which already ate ..., was full

$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$

(* elementwise)

If $\Gamma_u = 1 \longrightarrow$ update

$\Gamma_u = 0 \longrightarrow$ don't update



Full GRU

$\tilde{c}^{\langle t \rangle} = g(W_c [\Gamma_r * c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c)$ ← Gate tells relevance of $c^{\langle t-1 \rangle}$

$\Gamma_u = \sigma(W_u [c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$

$\Gamma_r = \sigma(W_r [c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_r)$

$c^{\langle t \rangle} = \Gamma_u \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$

LSTM (Long Short Term Memory)

$\tilde{c}^{\langle t \rangle} = \tanh(W_c [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$

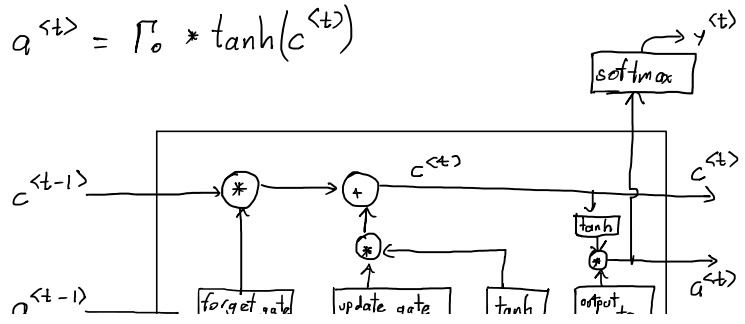
$\Gamma_u = \sigma(W_u [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$

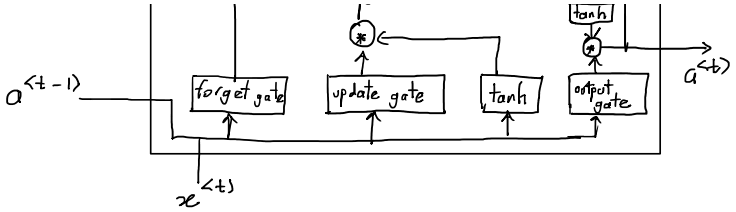
$\Gamma_f = \sigma(W_f [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f)$ ← $c^{\langle t-1 \rangle}$ peephole connection

$\Gamma_o = \sigma(W_o [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o)$

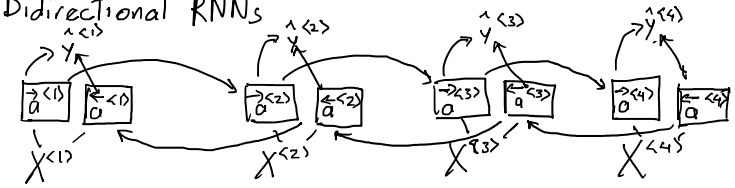
$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + \Gamma_f * c^{\langle t-1 \rangle}$

$a^{\langle t \rangle} = \Gamma_o * \tanh(c^{\langle t \rangle})$





Bidirectional RNNs



RNN
GRU
LSTM

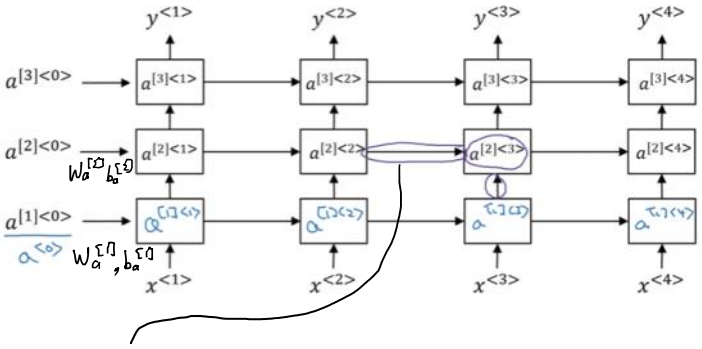
Acyclic graph

BRNN w/ LSTM blocks

→ Very commonly used

→ Disadvantage → can't be used for real time NLP applications

Deep RNNs



$$a^{[2]<3>} = g(W_a^{[2]}[a^{[2]<2>}, a^{[1]<3>}] + b_a^{[2]})$$

→ Can comprise RNNs, GRUs, LSTM

→ Can have a bidirectional variant as well

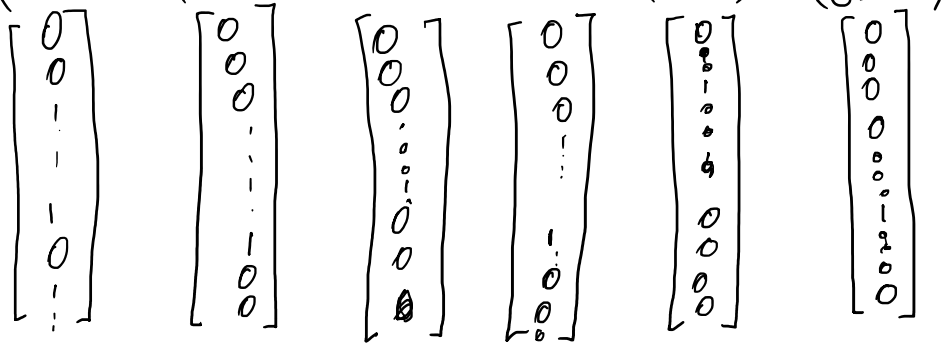
NLP and Word Embeddings

07 October 2021 12:55

$V = [a, aaron, \ldots\ldots zulu, \langle UNK \rangle]$

$|V| = 10{,}000$

Man (5391) Woman (9853) King (4914) Queen (7157) Apple (456) Orange (6257)

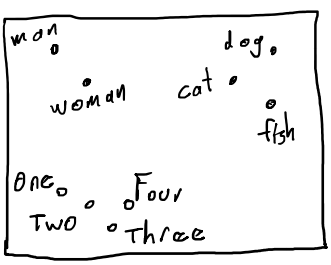


5391

I want a glass of orange juice

I want a glass of apple ——

	Man	Woman	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age						
Food						
⋮						
size						
cost						

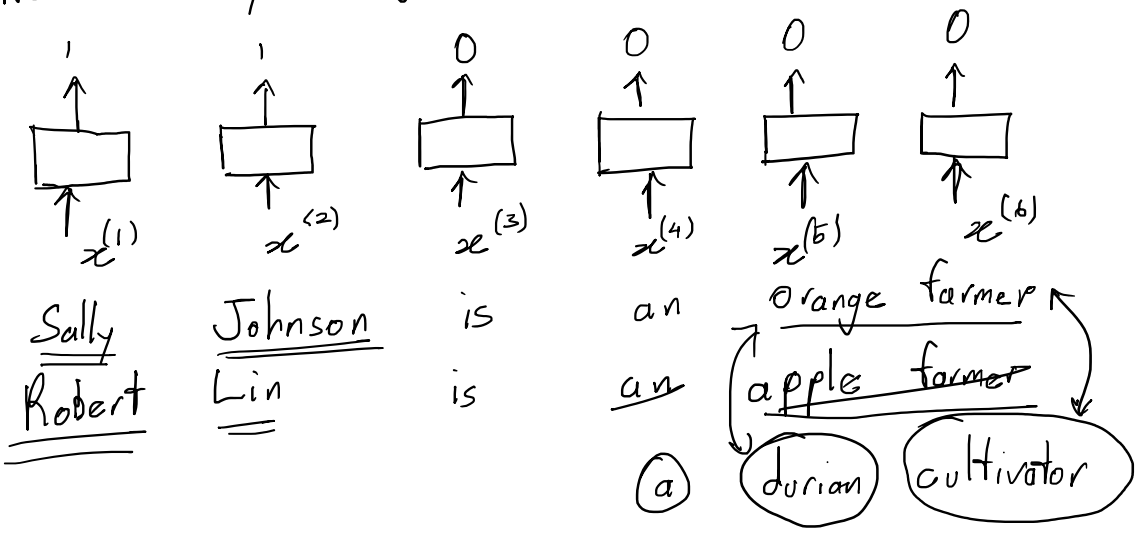


300D → 2D

t-SNE

Named Entity Recognition example

Named Entity Recognition example



→ 1 Billion words – 100 Billion words

→ 100K words

→ Use BRNNs

Transfer Learning & word embeddings

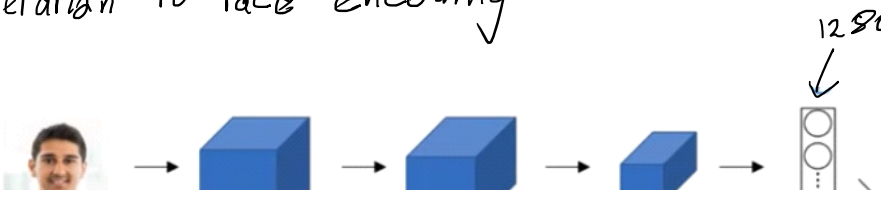
1) Learn word embeddings from large text corpus (1 - 100 B words)

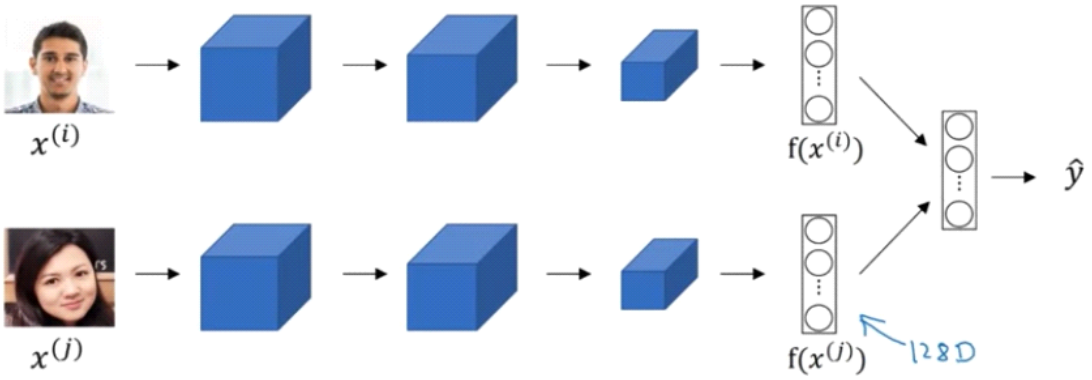
   Or download pre-trained word embeddings online

2) Transfer embedding to new task with smaller training set (say 100K words)

3) Optional: Continue to fine tune word embeddings with new data

Relation to face encoding





$|V| = 10000$

$\hookrightarrow e_1, e_2, \ldots, e_{10000}$

Analogies

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	−1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

$e_{5391} = e_{man}$ $\quad e_{woman}$

Man → Woman as King → Queen

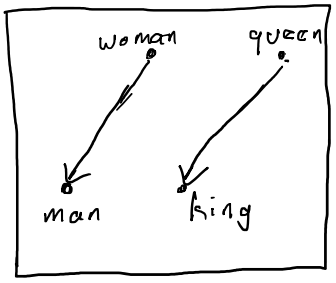
$$e_{man} - e_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$e_{king} - e_{queen} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$e_{man} - e_{woman} = e_{king} - e_?$ → queen

Analogies using word vectors

$e_{man} - e_{woman} \approx e_{king} - e_?$



t-SNE → 300 Dim → 2D

non linear mapping

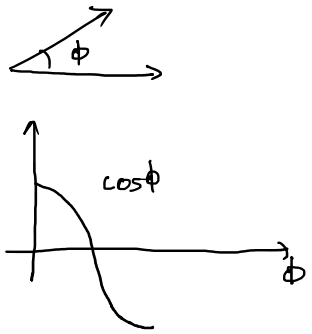
$e_{man} - e_{woman} \approx e_{king} - e_?$

Find word w: $\arg\max_w sim(e_w, e_{king} - e_{man} + e_{woman})$

Cosine similarity

$sim(e_w, e_{king} - e_{man} + e_{woman})$

$$sim(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$

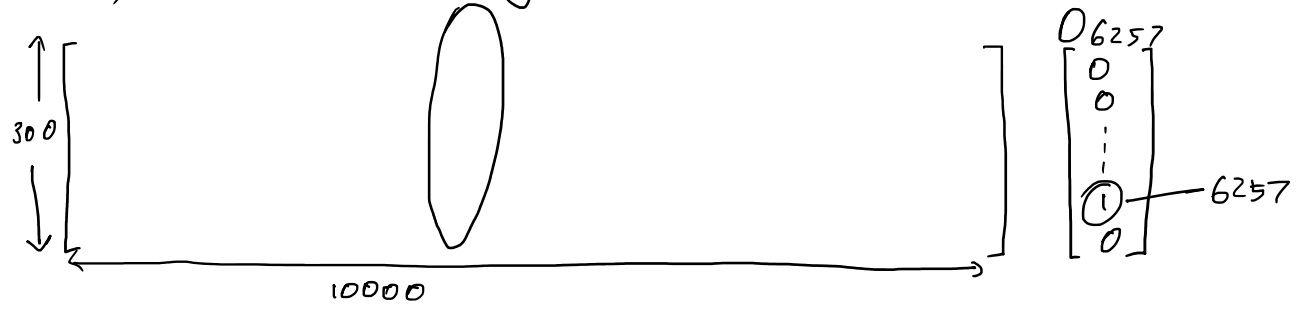


$\|u - v\|^2$

Man:Woman as Boy:Girl
Ottawa:Canada as Nairobi:Kenya
Big:Bigger as Tall:Taller
Yen:Japan as Ruble:Russia

Embedding Matrix

a, aaron .... orange ... zulu <unk>

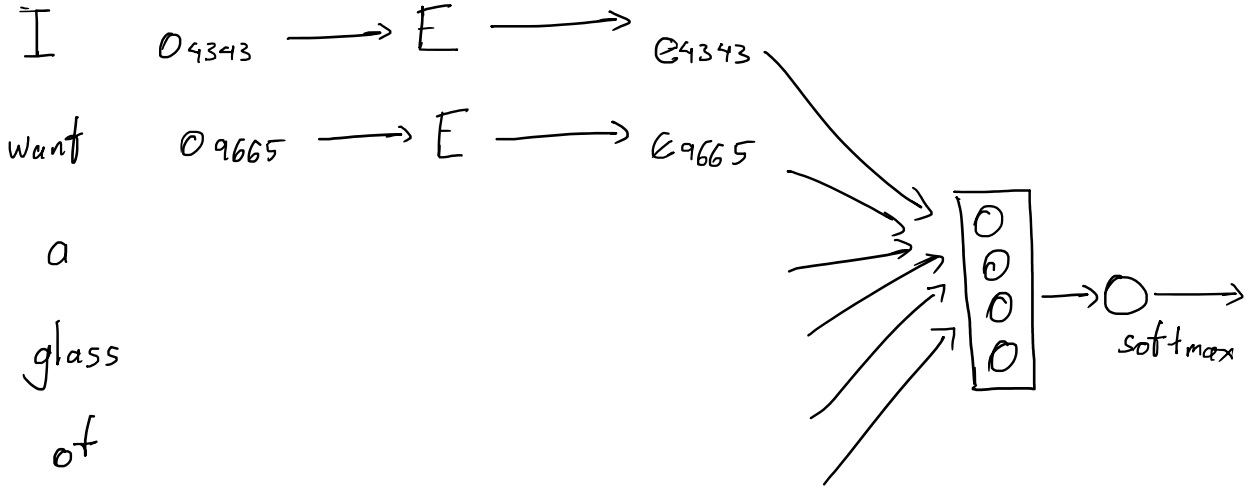


$E \cdot O_{6257} = e_{6257}$

(300, 10K) (10K, 1) (300, 1)

Learning Word Embeddings

I want a glass of orange ——— .
4343 9665 1 3852 6163 6257



I want a glass of orange juice to go along with my cereal

Context: Last 4 words

4 words on the left & right

Last 1 word — orange ?

Nearby 1 word — glass .... ?

skip gram

Word2Vec

Skip-Grams

I want a glass of orange juice to go along with my cereal

Context Target

orange juice

orange glass

orange my

Model

Vocab size = 10000

Context c('orange') 6257 $\longrightarrow$ Target t ('juice') 4834

$O_c \longrightarrow E \longrightarrow e_c \longrightarrow o \xrightarrow{softmax} \hat{y}$

$e_c = E O_c$

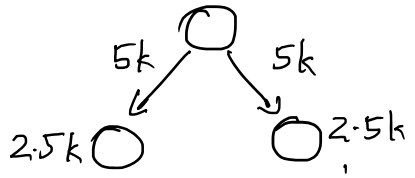
Softmax $p(t|C) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta}}$ θ_t = parameter associated with output t

$L(\hat{y}, y) = -\sum_{i=1}^{10000} y_i \log \hat{y}_i$

Problem

→ Computational Speed (sum over 10000 vectors)

Heirarchical Softmax



How to sample context c?

the, of, a, and, to → frequent

$c \rightarrow t \Rightarrow$ very frequent

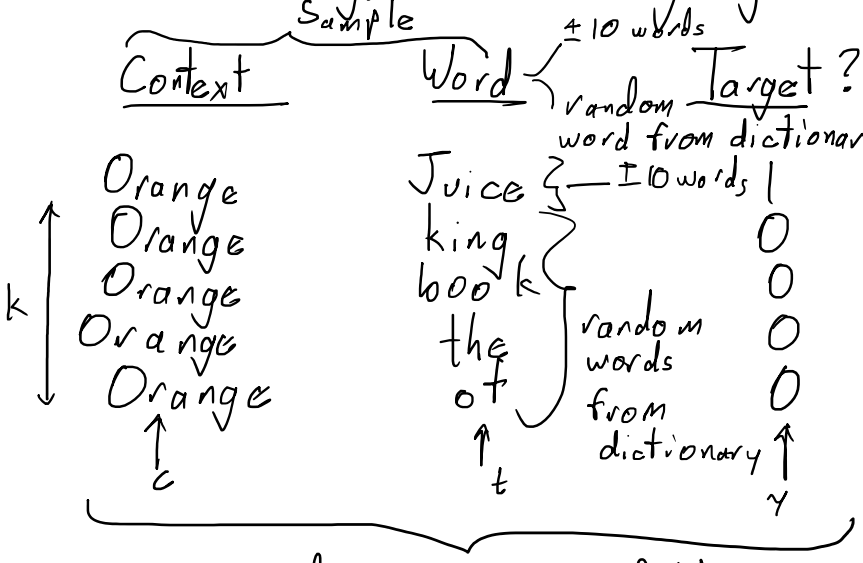
orange, apple, durian

There are ways to sample the sentence in order

to learn mapping while not spending most of the time in learning mapping for most frequent words

Negative Sampling

I want a glass of orange juice to go along with my cereal

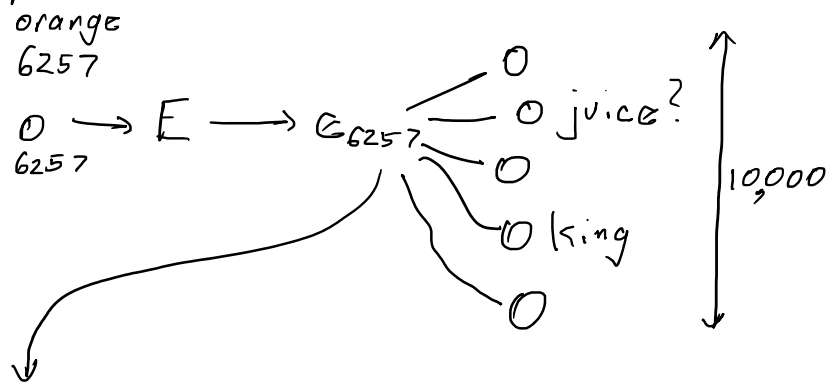


Supervised Learning Problem

$X \longrightarrow y$

$k = 5-20$ smaller datasets
$k = 2-5$ larger datasets

$P(y=1 \mid c,t) = \sigma(\theta_t^T e_c)$



Instead of training all 10,000, train randomly chosen 5 (along with actual target)

How to choose negative examples

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{10000} f(w_i)^{3/4}}$$

frequency of word in training corpus

$$f(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10000} f(w_j)^{3/4}}$$

GloVe Word Vectors
↳ Global Vectors for word representation

I want a glass of orange juice to go along with my cereal

X_{ij} ⇒ # times j (t) appears in the context of i (c)

$X_{ij} = X_{ji}$

$$\text{minimize} \sum_{i=1}^{10000} \sum_{j=1}^{10000} f(X_{ij}) (\theta_i^T e_j + b_i + b_j - \log X_{ij})^2$$

$\theta_i^T e_j$: t for θ_i, c for e_j; $\log X_{ij}$: $\theta_t^T e_c$

$f(X_{ij})$: weighting term

$f(X_{ij}) = 0$ if $X_{ij} = 0$ ("0 log 0" = 0)

↳ this, is, of, a, ... ← common
durian ← uncommon

θ_i, e_j are symmetric

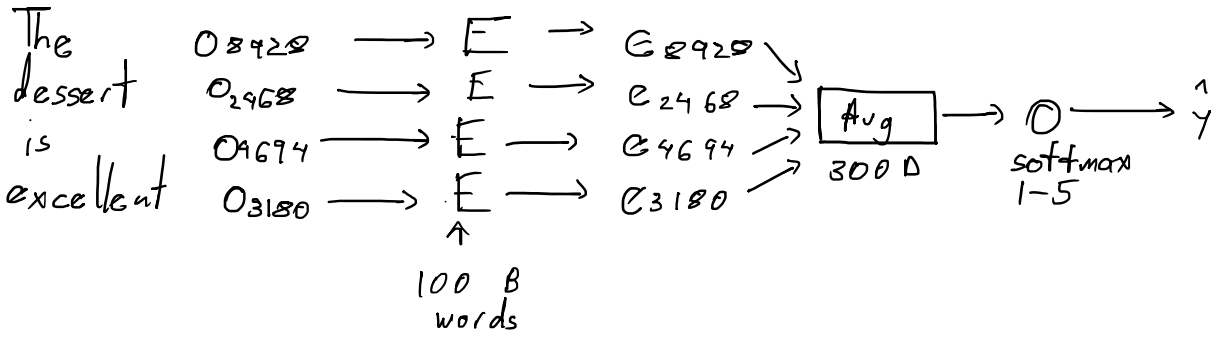
$$e_w^{final} = \frac{e_w + \theta_w}{2}$$

Sentiment Classification problem

x	y
The dessert is excellent.	★★★★☆
Service was quite slow.	★★☆☆☆
Good for a quick meal, but nothing special.	★★★☆☆
Completely lacking in good taste, good service, and good ambience.	★☆☆☆☆

10,000 → 1,00,000 words

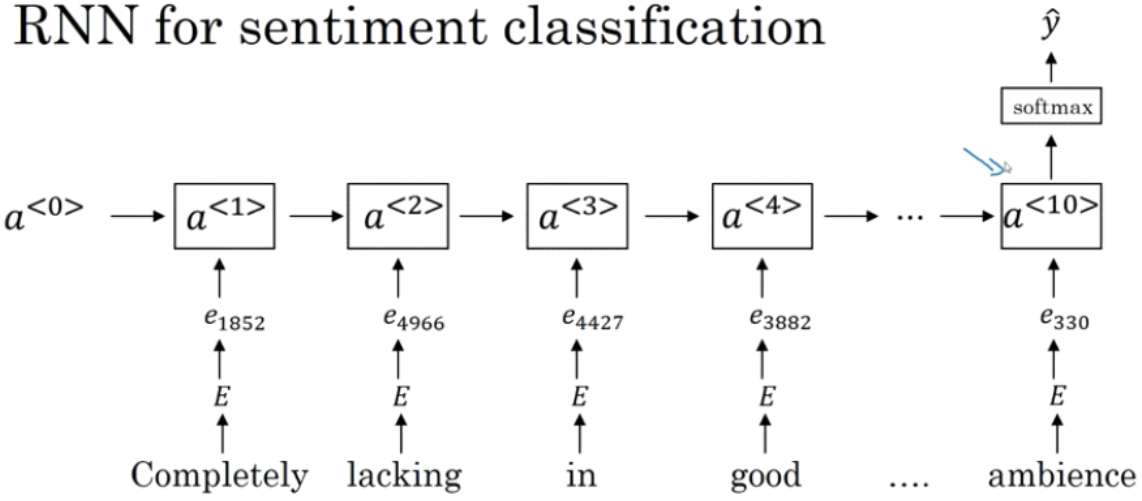
The dessert is excellent
8928 2468 4694 3180



Averages (kind of) the meaning of words

Cons
Ignores word order
eg ↳ Completely lacking in good taste, good service and good ambience

Instedd of Averaging, use an RNN !!



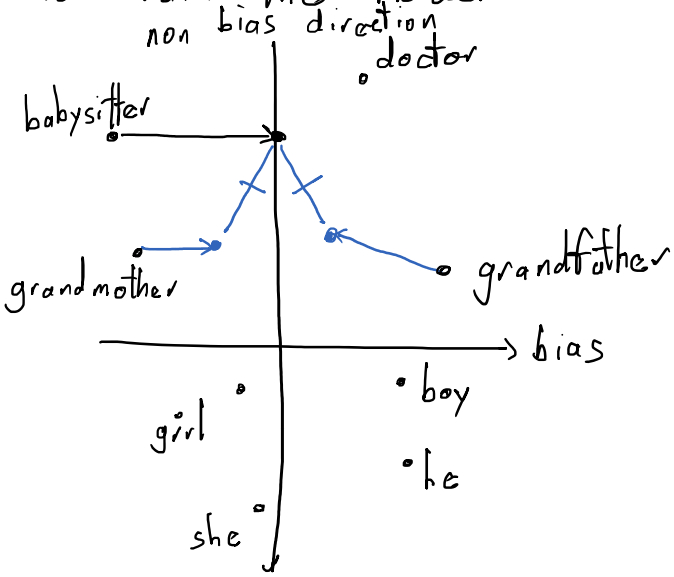
Debiasing word embeddings

Man : Woman as King : Queen

Man : Computer Programmer as Woman : Homemaker X

Father : Doctor as Mother : Nurse X

Word embeddings can reflect gender, ethnicity, age sexual orientation, and other biases of the text used to train the model



1) Identify bias direction

$e_{he} - e_{she}$
$e_{male} - e_{female}$
⋮
→ average / SVD

↓ Linear Classifier

2) Neutralize : For every word that is not definitional, project to get rid of bias

→ beard

3) Equalize pairs

→ Move grandmom & grandpa to locations that are equidistant from the non bias axis

Sequence to Sequence Architectures

08 October 2021 12:45

Basic Models

Jane visite l'Afrique en septembre

Jane is visiting Africa in September

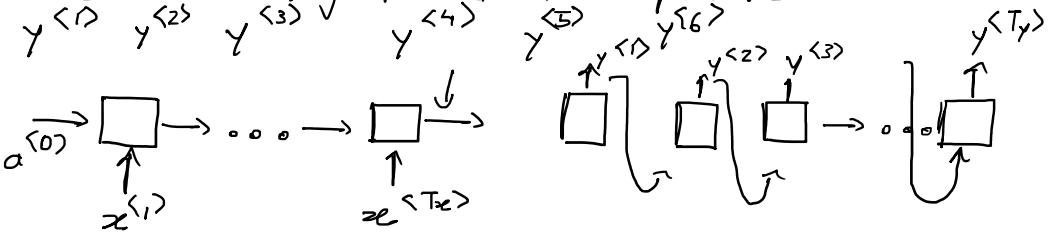
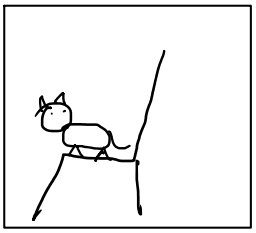
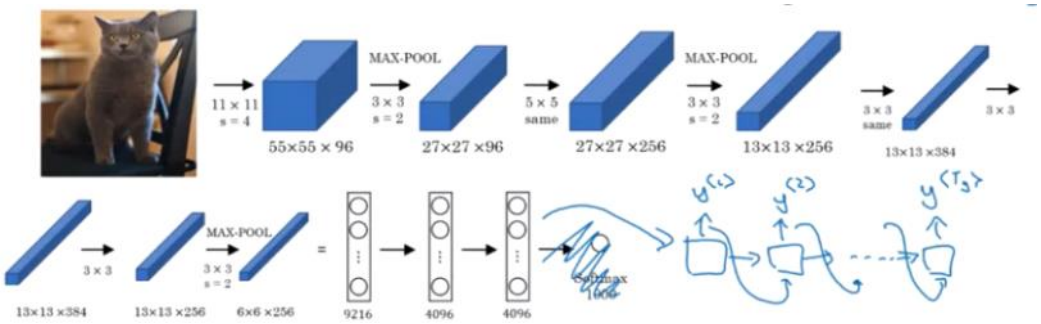


Image Captioning

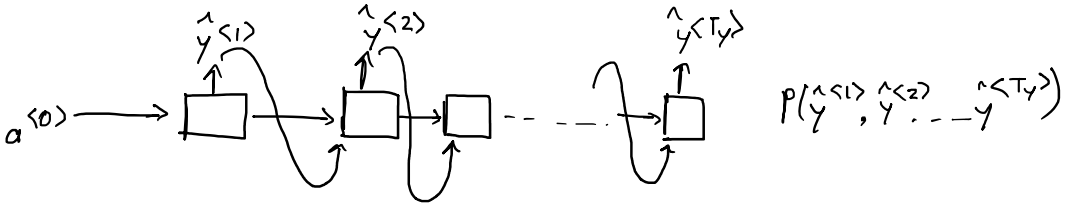


A cat sitting on a chair

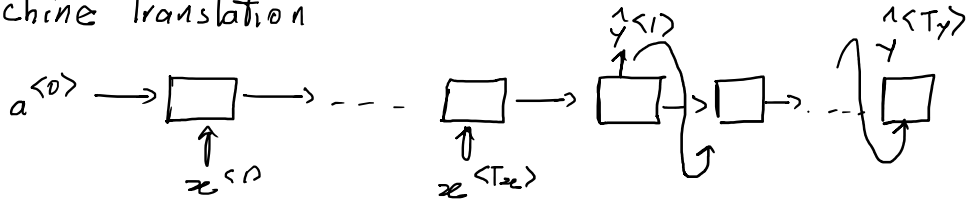


Picking the most likely Sentence

Language model:



Machine Translation



Conditional Language Model

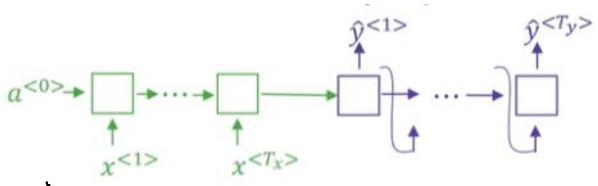
↳ $P(y^{<1>} \dots y^{<T_y>} | x^{<1>} \dots x^{<T_x>})$

Jane visite l'Afrique en septembre $P(y^{<1>}, y^{<2>} \dots y^{<T_y>} | x)$

→ Jane is visiting Africa in September
→ Jane is going to be visiting Africa in September
→ In September, Jane will visit Africa

$$\arg\max_{y^{<1>}, \dots y^{<T_y>}} P(y^{<1>}, y^{<2>} \dots y^{<T_y>} | x)$$

Why not a greedy search?



↑ Picks best words individually

1) Jane is visiting Africa in September ← better sentence

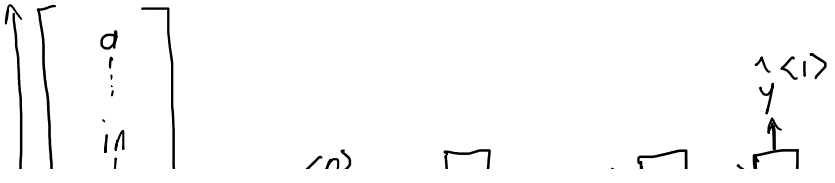
2) Jane is going to be visiting Africa in September

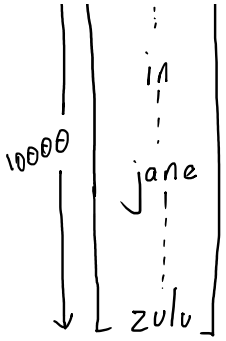
$P(2|X) > P(1|x)$

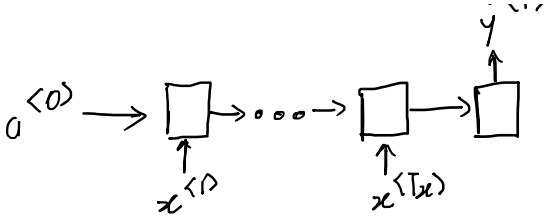
↑ going appears more frequently in English than visiting

Beam Search

Step 1

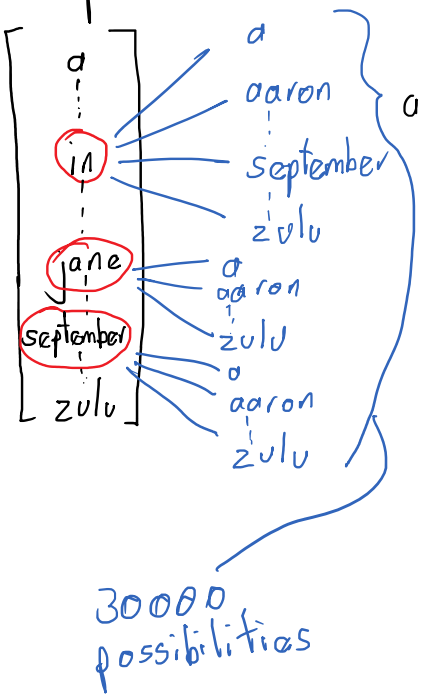


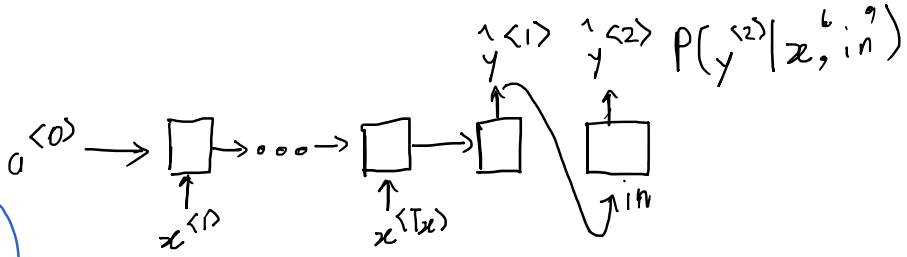




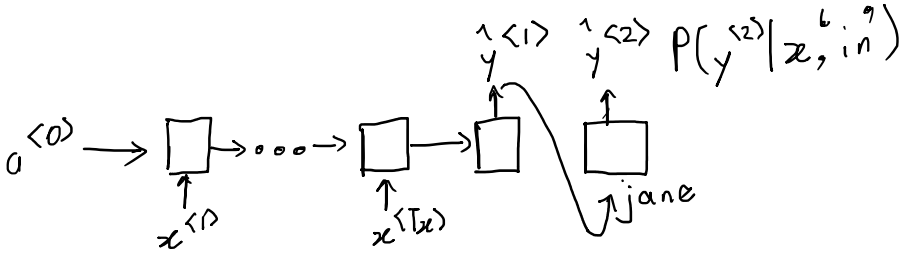
$P(y^{<1>} | x)$

Step 2

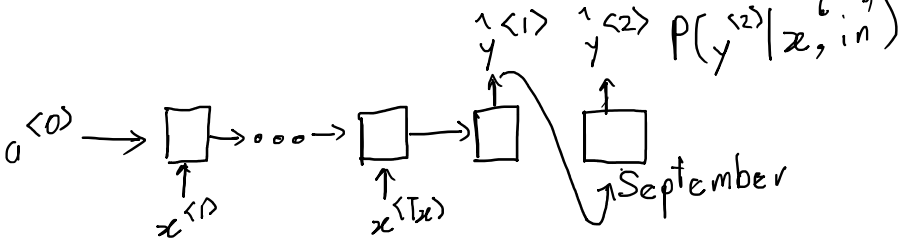




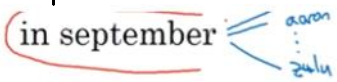
$P(y^{<1>}, y^{<2>} | X) = P(y^{<1>} | x) P(y^{<2>} | x, y^{<1>})$

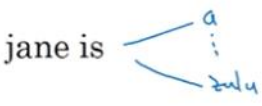


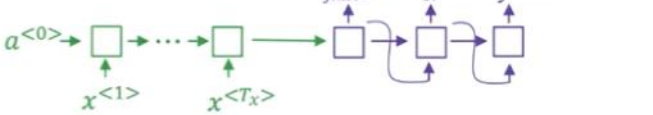
$P(y^{<2>} | X, \text{"jane"})$

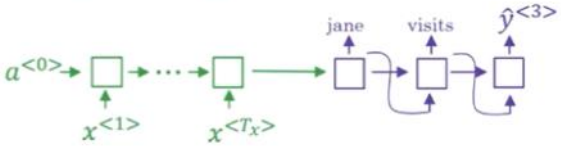


Step 3









$P(y^{<1>}, y^{<2>} | x)$

Length Normalization

$$\arg\max_{y} \prod_{t=1}^{T_y} P(y^{\langle t \rangle} | x, y^{\langle 1 \rangle}, \ldots y^{\langle t-1 \rangle})$$

↳ Can result in numerical underflow

$$\arg\max \sum_{y=1}^{T_y} \log P(y^{\langle t \rangle} | x, y^{\langle 1 \rangle} \ldots . y^{\langle t-1 \rangle})$$

Longer Sentence has less probability than short sentence

To fix this,

$$\frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{\langle t \rangle} | x, y^{\langle 1 \rangle} \_ \_ y^{\langle t-1 \rangle})$$

$\alpha = 0.7$ ($\alpha = 1$, $\alpha = 0$)

large B: better result, slower
small B: worse result, faster
} Beam width B

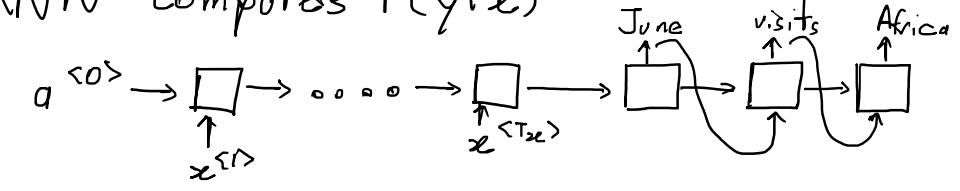
Unlike exact search algorithms like BFS (Breadth First Search) or DFS (Depth First Search), Beam Search runs faster but is not guaranteed to find exact maximum for $\arg\max_{y} P(y|x)$

Error Analysis in Beam Search

Jane visite l'Afrique en septembre

Human: Jane visits Africa in September (y^*)
Algo: Jane visited Africa in September ($\hat{y}$)

RNN computes $P(y|x)$



Case 1: $P(y^*|x) > P(\hat{y}|x)$

Beam Search chose $\hat{y}$, But y^* attains higher $P(y|x)$

Conclusion: Beam search is at fault

Case 2: $P(y^*|x) \leq P(\hat{y}|x)$

y^* is a better translation than $\hat{y}$. But RNN predicted $P(y^*|x) < P(\hat{y}|x)$

Conclusion: RNN is at fault

Error analysis process

Human	Algorithm	$P(y^*\|x)$	$P(\hat{y}\|x)$	At fault?
Jane visits Africa in September.	Jane visited Africa last September.	2×10^{-10}	1×10^{-10}	B
...	...	—	—	R
...	...	—	—	B
				R
				R
				⋮

Figures out what fraction of errors are due to beam search vs RNN model

Bleu Score

French: Le chat est sur le tapis

Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

BLEU

→ Bilingual Evaluation Understudy

MT output: the the the the the the the.

Precision: $\frac{7}{7}$ → Not useful

Modified precision: $\frac{2}{7}$ → max no of times it appears on the reference

Bleu score on bigrams

French: Le chat est sur le tapis

Reference 1: The cat is on the mat.
Reference 2: There is a cat on the mat.

MT output: The cat the cat on the mat.

	Count	Count_{clip}
the cat	2	1
cat the	1	0
cat on	1	1
on the	1	1
the mat	1	1

Modified precision on bigrams = $\frac{4}{6}$

$$p_1 = \frac{\sum_{\text{unigrams} \in \hat{y}} (\text{Count}_{clip}(\text{unigram}))}{\sum_{\text{unigrams} \in \hat{y}} \text{Count}(\text{unigram})}$$

n-gram:

$$p_n = \frac{\sum_{\text{n-grams} \in \hat{y}} \text{Count}_{clip}(\text{n-gram})}{\sum_{\text{n-grams} \in \hat{y}} \text{Count}(\text{n-gram})}$$

BLEU

$p_n \Rightarrow$ Bleu score on n-grams only.

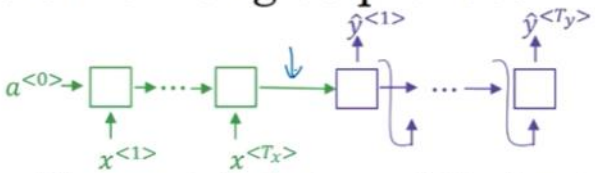
Combined Bleu score = B.P. $e^{\left(\frac{1}{4}\sum_{n=1} p_n\right)}$

B.P. → Brevity Penalty — adjustment factor

$$BP \begin{cases} 1 & \text{if MT\_output\_length} > \text{reference\_output\_length} \\ e^{(1-\text{reference\_output\_length}/\text{MT\_output\_length})} & \text{otherwise} \end{cases}$$

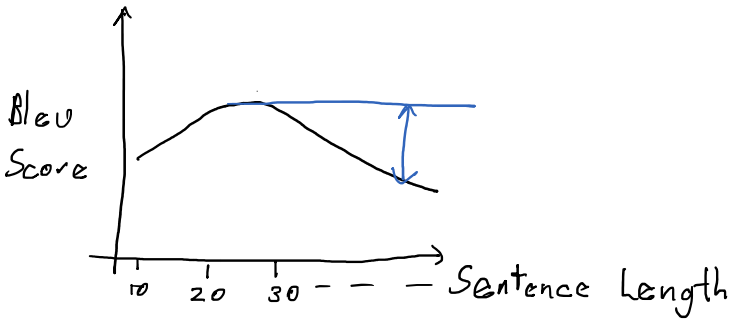
Attention Model Intuition

The problem of long sequences

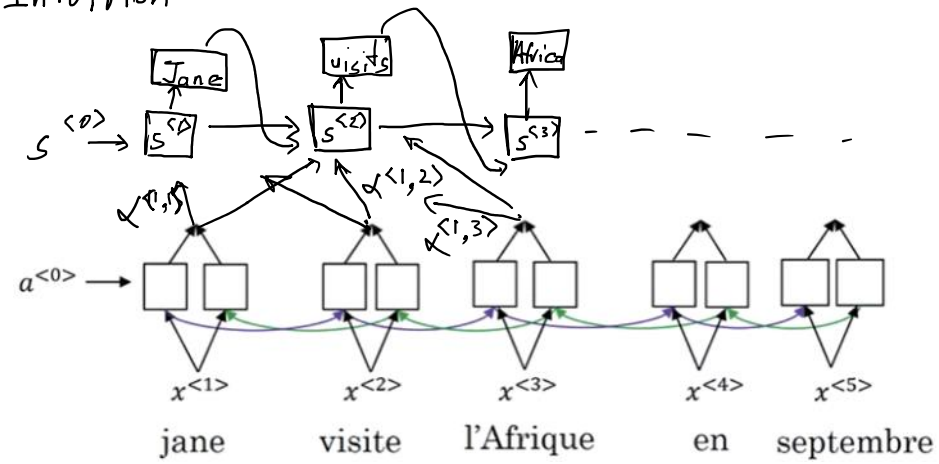


Jane s'est rendue en Afrique en septembre dernier, a apprécié la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y aller aussi.

Jane went to Africa last September, and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was, and is tempting me to go too.

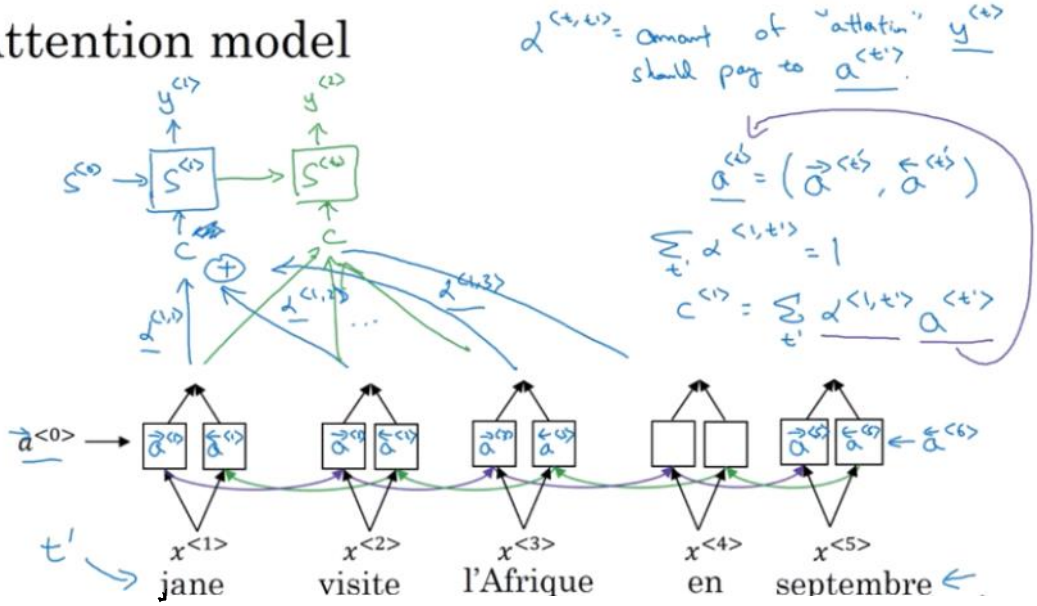


Intuition



Attention Model

Attention model



$\alpha^{<t,t'>}$ = amount of "attention" $y^{<t>}$ should pay to $a^{<t'>}$

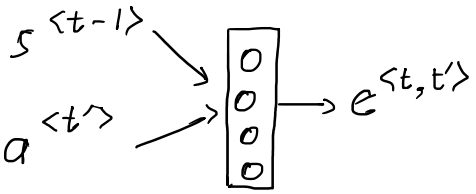
$a^{<t'>} = (\vec{a}^{<t'>}, \overleftarrow{a}^{<t'>})$

$\sum_{t'} \alpha^{<1,t'>} = 1$

$c^{<1>} = \sum_{t'} \alpha^{<1,t'>} a^{<t'>}$

$\alpha^{<t,t'>}$ ⇒ Amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$

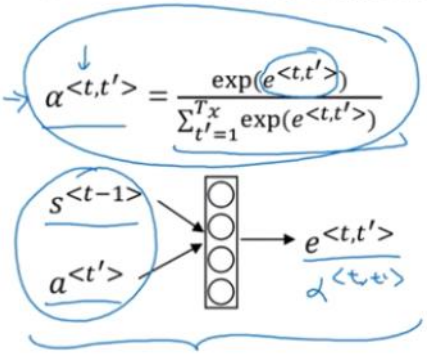


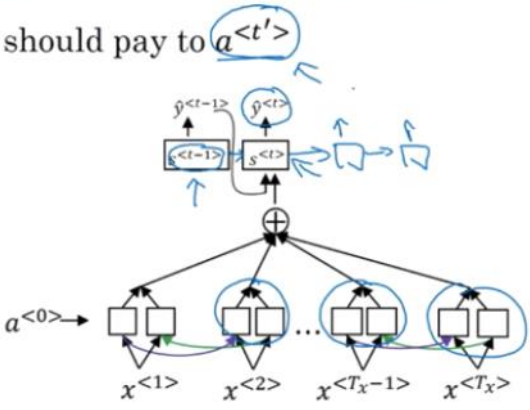
Computing attention $\alpha^{<t,t'>}$

T_x T_y

$\alpha^{<t,t'>}$ = amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$



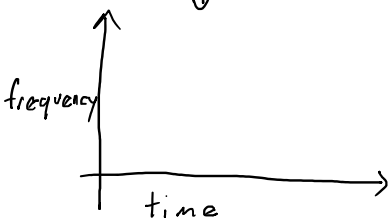


Speech Recognition Problem

x audio clip ⟶ y

⟶ "the quick brown fox"

→ "the quick brown fox"



Color shows amplitude of frequency at time t

Spectrogram

phoneme : Basic units of sound

↓

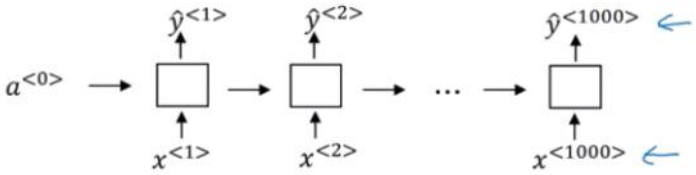
Hand engineered features

Academic → 300h
3000h

Industry → 10,000h

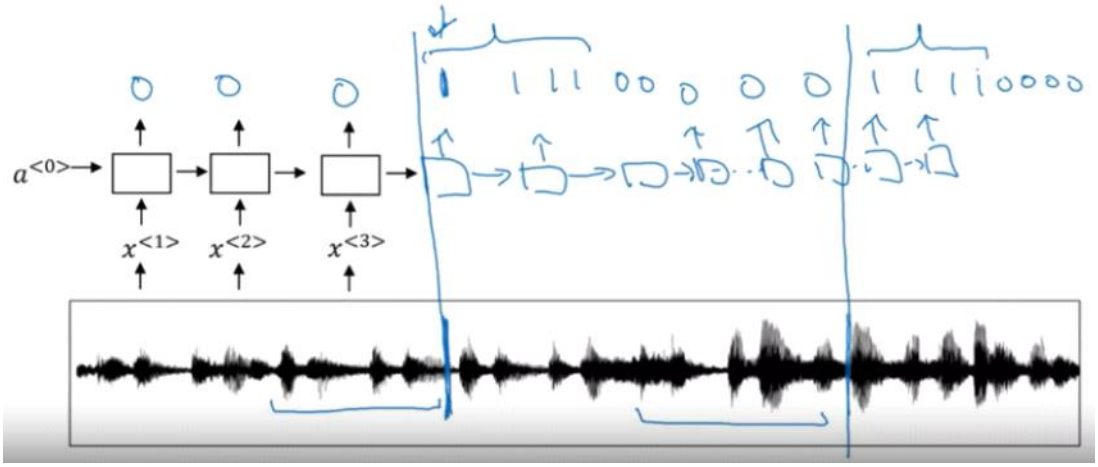
CTC cost for speech recognition

(Connectionist Temporal Classification)



Basic Rule : Collapse repeated characters not separated by "blank"

Trigger word detection algorithm

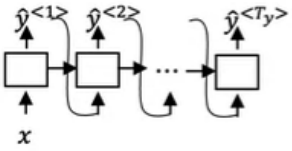

$a^{<0>}$
$x^{<1>}$
$x^{<2>}$
$x^{<3>}$

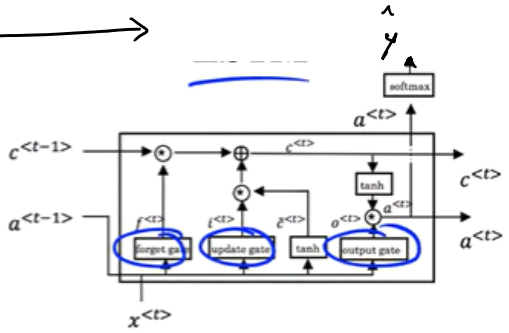
Transformer Networks

08 October 2021 16:48

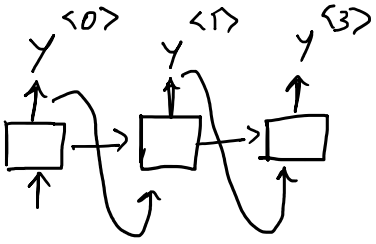
RNN → GRU → LSTM

Increase complexity

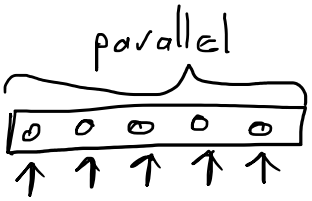




- Attention + CNN



parallel



$A^{<1>}$ $A^{<2>}$ $A^{<3>}$ $A^{<4>}$ $A^{<5>}$

- Self Attention
- Multi-Head Attention

Self Attention

$A(q, K, V)$ = attention based vector representation of word

→ Calculate for each word $A^{<1>}$ ... $A^{<5>}$

RNN Attention

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$

Transformers Attention

$$A(q,K,V) = \sum_i \frac{\exp(q \cdot k^{<i>})}{\sum_j \exp(q \cdot k^{<j>})} v^{<i>}$$

$A^{<3>}$

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$

Jane visite l'Afrique en septembre

→ $q^{<3>}$ $k^{<3>}$ $v^{<3>}$

Self Attention

Query
$q^{<1>}$
$q^{<2>}$
$q^{<3>}$
$q^{<4>}$
$q^{<5>}$
q

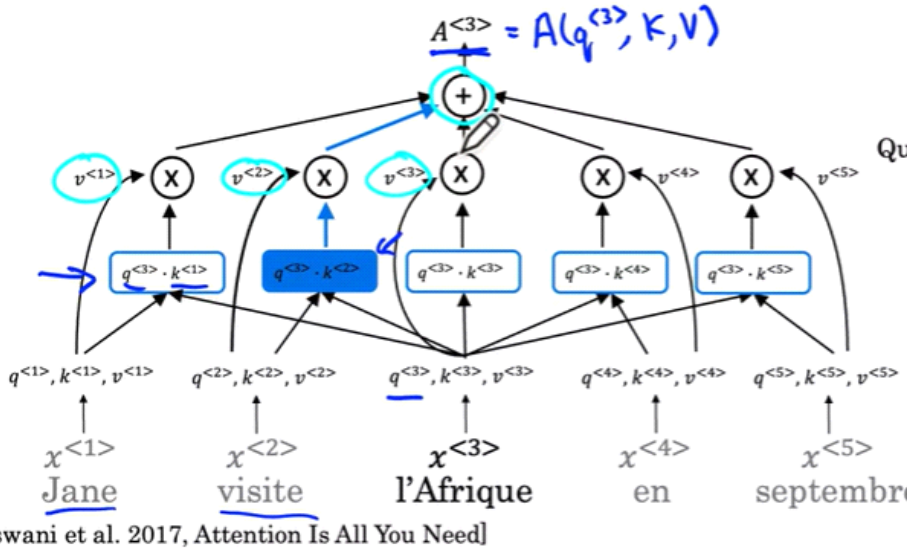
Key
$k^{<1>}$ person
$k^{<2>}$ action
$k^{<3>}$
$k^{<4>}$
$k^{<5>}$

Value
$v^{<1>}$
$v^{<2>}$
$v^{<3>}$
$v^{<4>}$
$v^{<5>}$

$q^{<3>} = W^{Q} x^{<3>}$ → whats happening there
$k^{<3>} = W^{K} x^{<3>}$
$v^{<3>} = W^{V} x^{<3>}$

Self-Attention

$$A(q, K, V) = \sum_i \frac{\exp(q \cdot k^{<i>})}{\sum_j \exp(q \cdot k^{<j>})} v^{<i>}$$



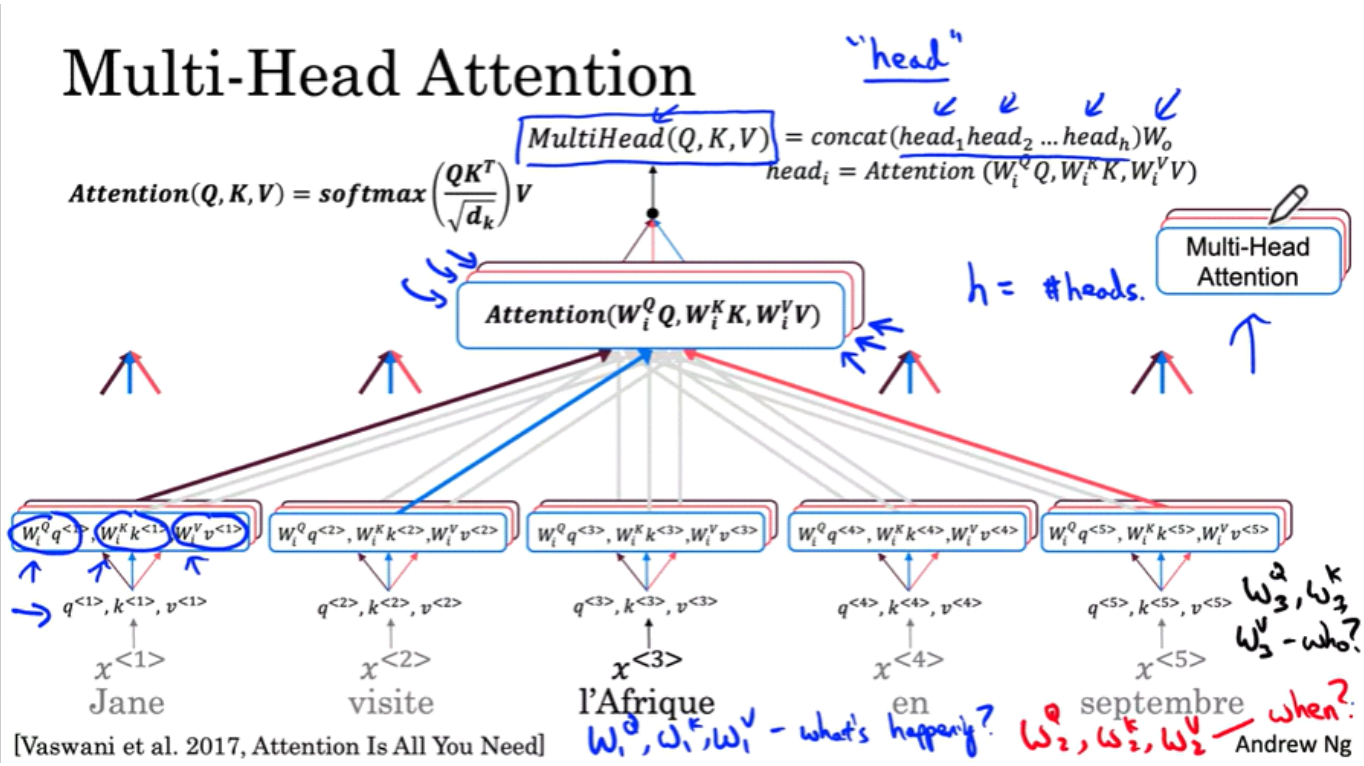
[Vaswani et al. 2017, Attention Is All You Need]

Query (Q)	Key (K)	Value (V)
$q^{<1>}$	$k^{<1>}$ person	$v^{<1>}$
$q^{<2>}$	$k^{<2>}$ action	$v^{<2>}$
$q^{<3>}$ what's happening there	$k^{<3>}$	$v^{<3>}$
$q^{<4>}$	$k^{<4>}$	$v^{<4>}$
$q^{<5>}$	$k^{<5>}$	$v^{<5>}$

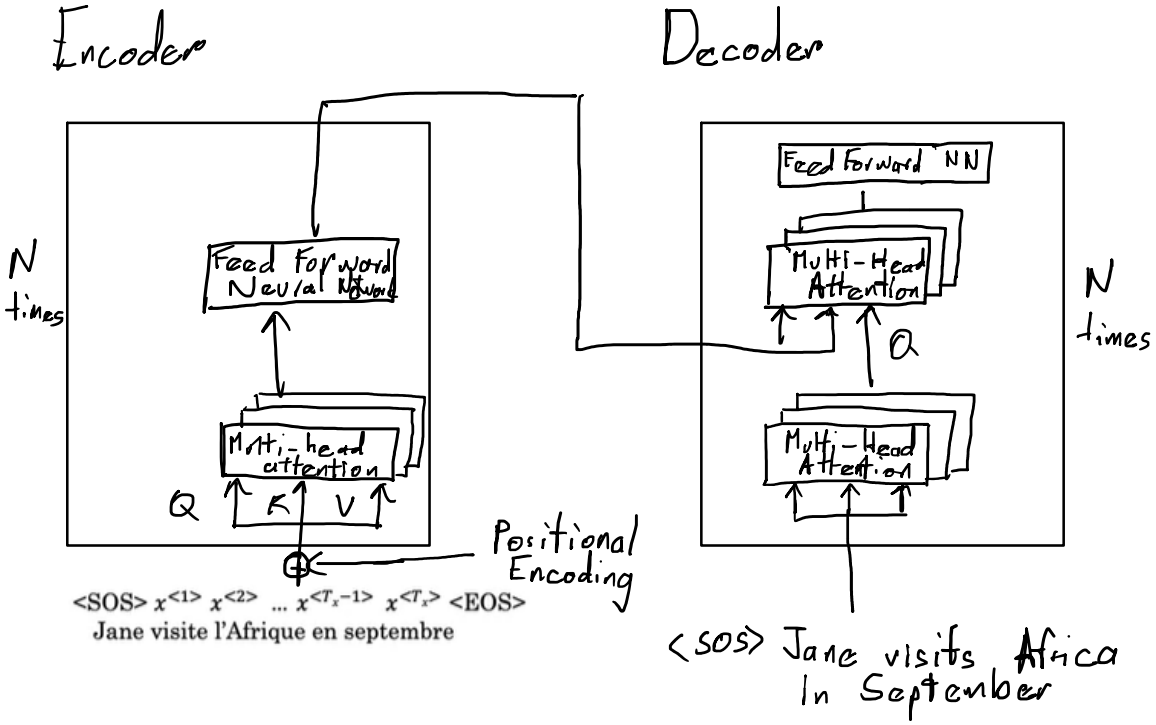
$q^{<3>} = W^{Q} x^{<3>}$
$k^{<3>} = W^{K} x^{<3>}$
$v^{<3>} = W^{V} x^{<3>}$

Andrew Ng

Multi-Head Attention



Transformer Network



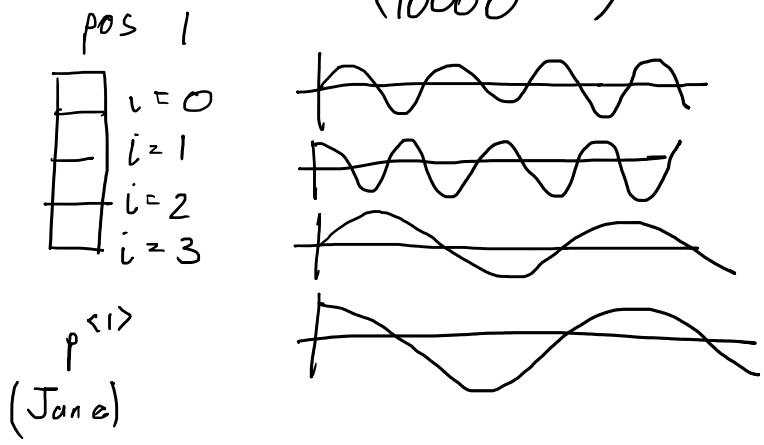
Positional Encoding

numerical positional

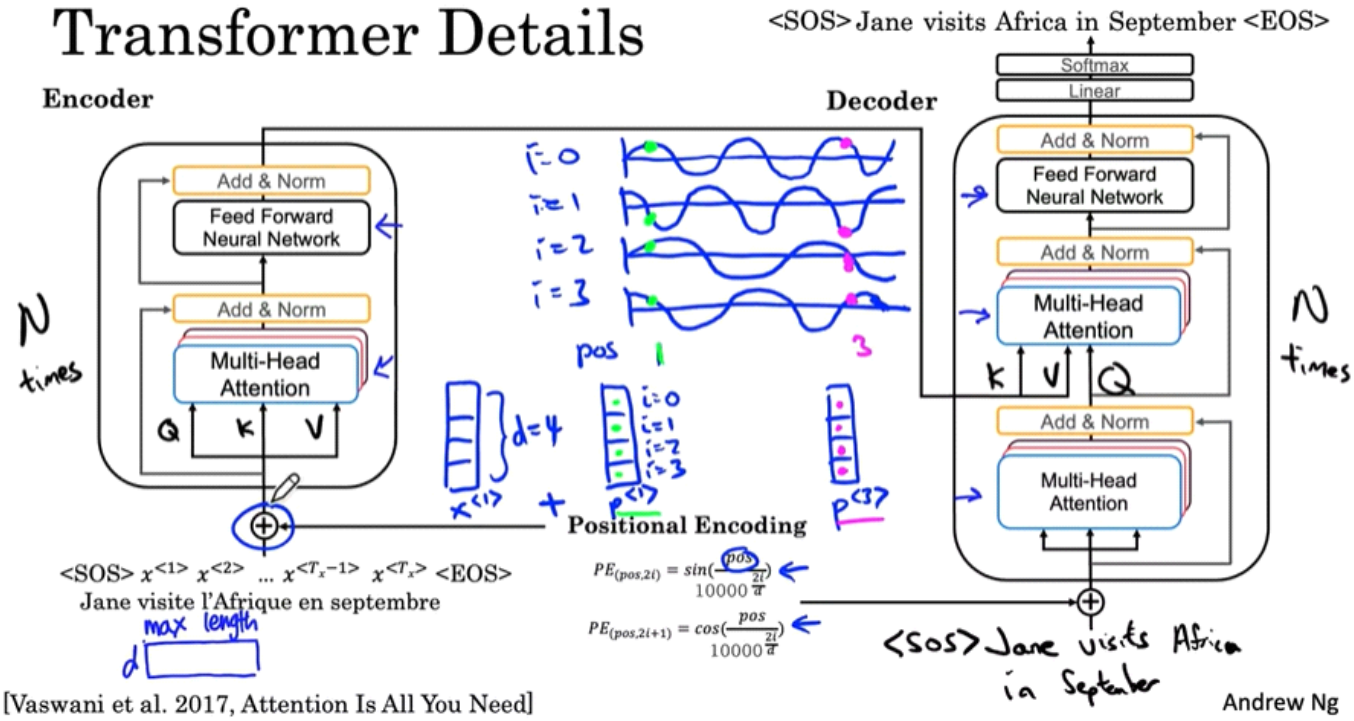
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$



Transformer Details



[Vaswani et al. 2017, Attention Is All You Need]

Andrew Ng

See Stanford Video on Youtube

Natural Language Processing

11 October 2021 11:32

How do we represent the meaning of a word?

signifier (symbol) ⇔ signified (idea or thing)
= denotation

Common solution :- use a word net

A thesaurus containing lists of synonyms & hypernyms ('is a' relationships)

Problems with resources like WordNet

→ Great as a resource but missing nuance

eg. 'proficient' is listed as a synonym for good

→ Missing new meanings of words

→ Subjective
→ Requires human labour to create & adapt
→ Can't compute word similarity

One hot vectors ⟶ localist representation

→ Orthogonal !!
→ No notion of similarity

Solutions:
- Build word similarity table / WordNets
- Learn to encode similarity in the vectors themselves

Representing words by their content

Distributional semantics — Know a word by the company it keeps

When word w appears in a text, its context is the set of words that appear nearby (within a fixed size window)

Use many contexts of w to build up a representation of w

Word vectors / Word embeddings / Word representations

↳ Dense vector for each word chosen

banking = $\begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$

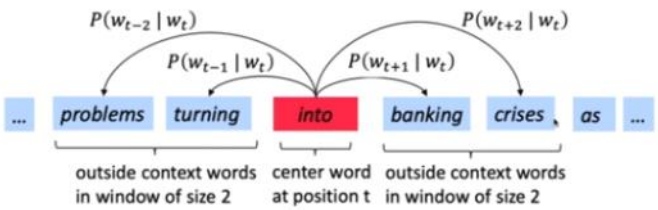
Word2Vec

Idea

- ↳ Large corpus of text
- ↳ Every word in a fixed vocabulary is represented by a vector
- ↳ Go through each position t in text which has center word c & context o
- ↳ Similarity of word vectors for c & o to calculate the probability of o given c
- ↳ Keep adjusting word vectors to maximize this probability

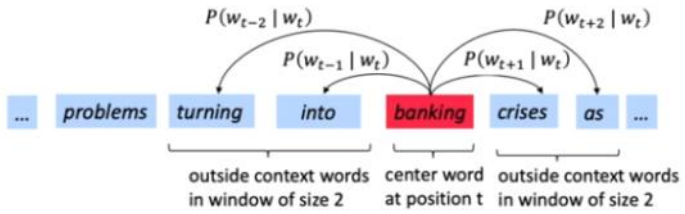
Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j.

Likelihood $= L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$

All variables to be optimized

Cost Fn

$$J(\theta) = -\frac{1}{T} \log L(\theta)$$

$$= -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m} \log P(w_{t+j} \mid w_t; \theta)$$

Minimize Cost Fn $\Leftrightarrow$ Maximizing predictive accuracy

$$J(\theta) = -\frac{1}{T} \sum \sum \log P(w_{t+j} \mid w_t; \theta)$$

How to calculate $P(w_{t+j} \mid w_t; \theta)$?

We will use two vectors per word w:

→ v_w when w is a center word

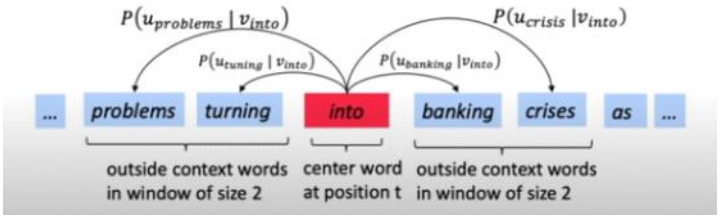
u_w when w is a context word

Then for a center word c with context words o:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{u \in w} \exp(u_w^T v_c)}$$

Word2Vec Overview with Vectors

- Example windows and process for computing $P(w_{t+j} \mid w_t)$
- $P(u_{problems} \mid v_{into})$ short for $P(problems \mid into\,; u_{problems}, v_{into}, \theta)$



$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in v} \exp(u_w^T v_c)}$$

Dot product compares the similarity of o & c

$u^T v = u \cdot v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

Normalize over entire vocabulary to give probability distribution

Example of the softmax function $\mathbb{R}^n \to \mathbb{R}^n$

Softmax maps arbitrary values x_i to a probability distribution p_i

To train the model: Compute all vector gradients

θ represents all model parameters in 1 long vector

In our case, with d dimensional vectors & V-many words:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

$$\begin{bmatrix} u_a \\ \vdots \\ u_{zebra} \end{bmatrix}$$

Remember — every word has 2 vectors

We optimize these parameters by walking down the gradient

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

$$= \frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) - \frac{\partial}{\partial v_c} \log \left(\sum_{w=1}^{V} \exp(u_w^T v_c)\right)$$

numerator denominator

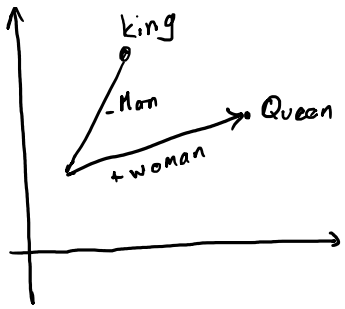
$$= \frac{\partial}{\partial v_c} u_o^T v_c - \frac{\partial}{\partial v_c} \log \left(\sum_{w=1}^{V} \exp(u_w^T v_c)\right)$$

$$= u_o - \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \sum_{w=1}^{V} \exp(u_w^T v_c) \cdot u_w$$

$$= u_o - \sum_{w=1}^{V} \frac{\exp(u_w^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} u_w$$

$$= u_o - \sum_{w=1}^{V} p(w|c) \cdot u_w$$

```
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')

from sklearn.manifold import TSNE
from sklearn.decompose import PCA

from gensim.test.utils import datapath, get_tmpfile
from gensim.models import KeyedVectors
from gensim.scripts.glove2word2vec import glove2word2vec

glove_file = datapath('/Users/manning/Corpora/GloVe/glove.6B.....')
word2vec_glove_file = get_tmpfile('glove.6B.100d.word2vec.txt')
glove2word2vec(glove_file, word2vec_glove_file)

model = KeyedVectors.load_word2vec_format(word2vec_glove_file)
model.most_similar('obama')
model.most_similar('banana')
```



Machine Translation (Statistical MT)

x (source) $\longrightarrow$ y (target)

Suppose French $\longrightarrow$ English

$\text{argmax}_y \; P(y|x)$

Bayes Rule

$\text{argmax}_y \; P(x|y)P(y)$

Translation Model: Models how words & phrases should be translated

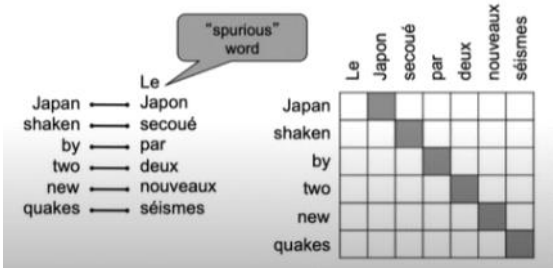
Language Model: How to write good English

How to learn translation model $P(x|y)$ from the parallel corpus

Break it down further

$P(x, a|y)$

$a \rightarrow$ alignment

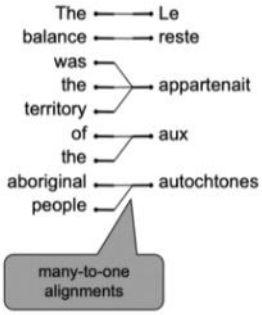


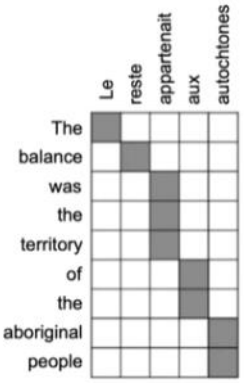
Alignment is complex

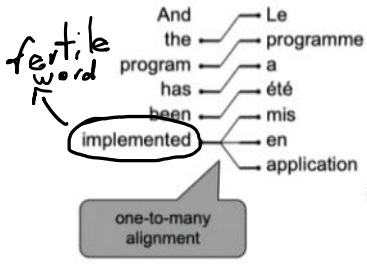
Alignment can be one-to-many

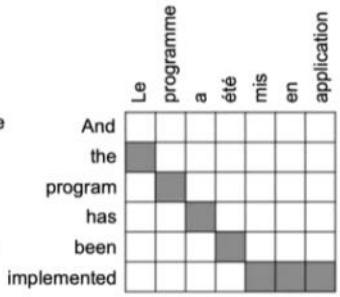
Alignment can be one-to-many

Alignment can be many-to-one

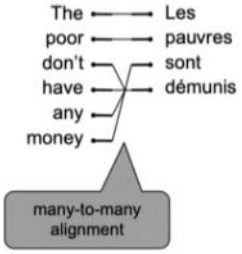


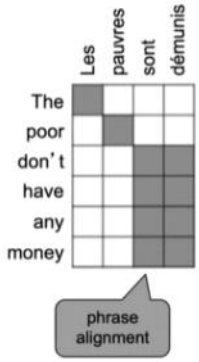






Alignment can be many-to-many (phrase-level)

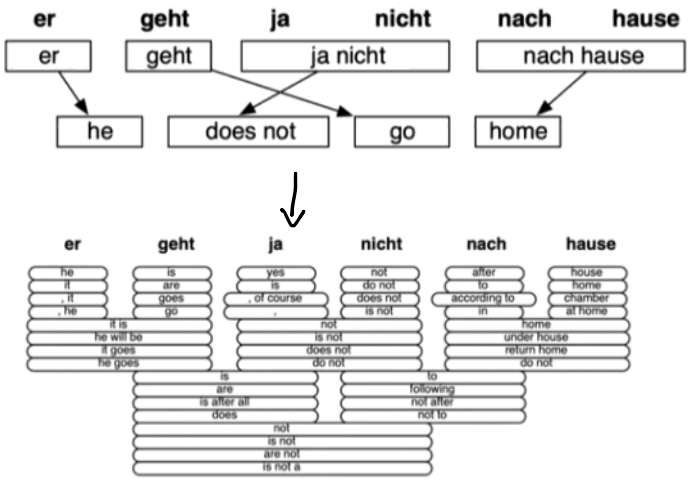




→ Enumerate every possible y & calculate the probability — too expensive!

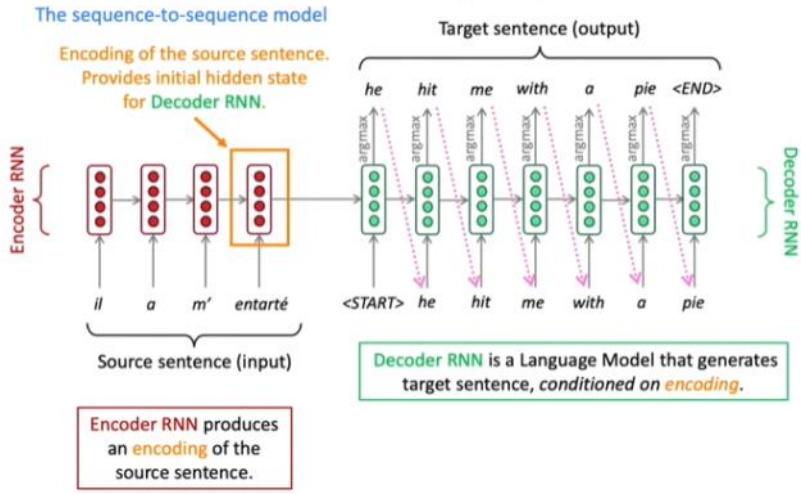
→ Use a heuristic search algorithm to search for the best translation, discarding hypotheses that are too low probability

→ Process is called decoding



Neural Machine Translation

← At test time



2 sets of word embeddings ← French, English

→ Summarisation (long text → short text)
→ Dialogue (previous utterance → next utterance)
→ Parsing
→ Code generation

Seq2Seq → Conditional language Model

- The sequence-to-sequence model is an example of a **Conditional Language Model**.
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are *also* conditioned on the source sentence *x*

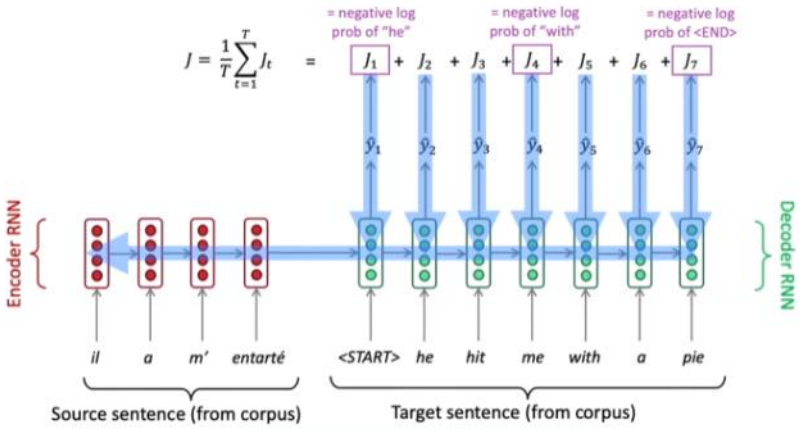
- NMT directly calculates $P(y|x)$:

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots\underbrace{P(y_T|y_1,\ldots,y_{T-1},x)}$$

Probability of next target word, given target words so far and source sentence *x*

How to train?
Get a big parallel corpus

Training a Neural Machine Translation system

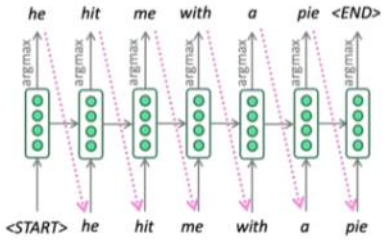


Seq2seq is optimized as a **single system.**
Backpropagation operates *"end-to-end"*.

27

Greedy Decoding

- We saw how to generate (or "decode") the target sentence by taking argmax on each step of the decoder



- This is greedy decoding (take most probable word on each step)
- **Problems with this method?**

→ expensive (can't be parallelised)
→ No way to undo decisions

Fix?

Beam search decoding

k → integer (beam size)
(5 - 10)

keep track of k most probable partial translations

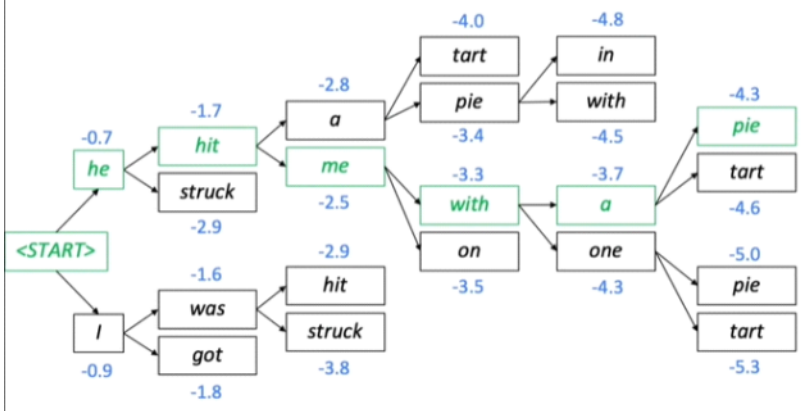
- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step

Beam search decoding: example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

Beam search decoding: stopping criterion

- In greedy decoding, usually we decode until the model produces a <END> token
  - For example: *<START> he hit me with a pie <END>*
- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.

Continue Beam Search till:

We reach timestep T (where T is some cutoff)

We accumulate n hypotheses

How to select top one?

- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

- Problem with this: longer hypotheses have lower scores

Fix: Normalize by length

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

Advantages of NMT

→ Better performance
  → More fluent
  → Better use of context

- Better performance
  - → More fluent
  - → Better use of context
  - → Better use of phrase similarities

- A single neural network to be optimized end-to-end
  - No subcomponents to be individually optimized

- Requires much less human engineering effort
  - No feature engineering
  - Same method for all language pairs

Disadvantages of NMT

Compared to SMT:

- NMT is less interpretable
  - Hard to debug

- NMT is difficult to control
  - For example, can't easily specify rules or guidelines for translation
  - Safety concerns!

How do we evaluate Machine Translation?

BLEU (**B**ilingual **E**valuation **U**nderstudy)

- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - *n*-gram precision (usually for 1, 2, 3 and 4-grams)
  - Plus a penalty for too-short system translations

- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - So a good translation can get a poor BLEU score because it has low *n*-gram overlap with the human translation ☹

Is MT solved?

Nope !!

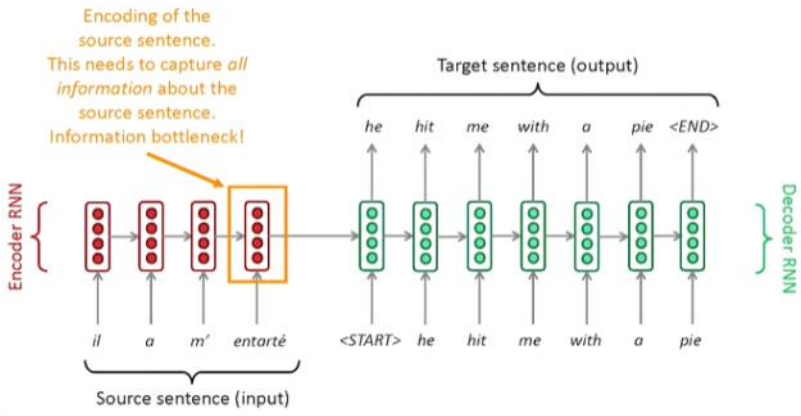
Many difficulties remain:

- Out-of-vocabulary words
- Domain mismatch between train and test data
- Maintaining context over longer text
- Low-resource language pairs

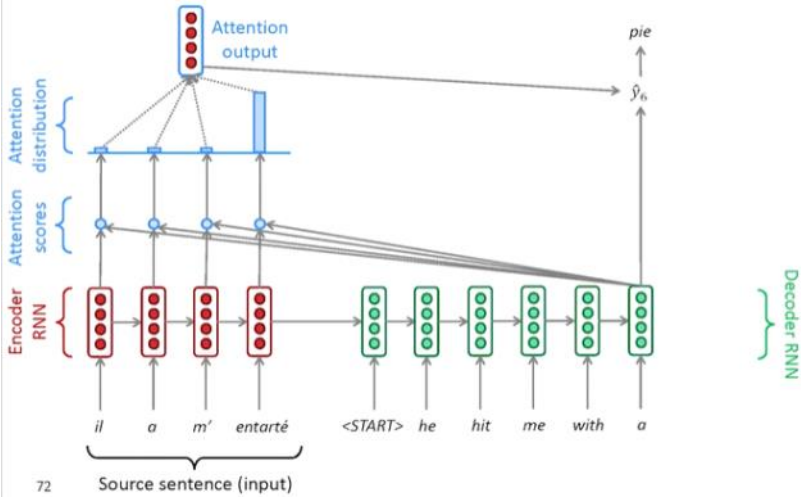
→ Common sense lacking

→ Common sense lacking
→ Picks up biases in the training data
→ Uninterpretable systems do strange things

Sequence-to-sequence: the bottleneck problem



Sequence-to-sequence with attention



72

Attention: in equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

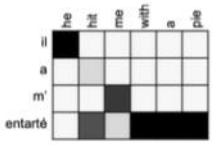
- Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[a_t; s_t] \in \mathbb{R}^{2h}$$

73

Attention is great

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself



Attention is a *general* Deep Learning technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

- More general definition of attention:
  - Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).

75

Attention is a *general* Deep Learning technique

More general definition of attention:
Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.

Intuition:

- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

Attention is a *general* Deep Learning technique

More general definition of attention:
Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.

Intuition:

- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

Attention variants

You'll think about the relative advantages/disadvantages of these in Assignment 4!

There are several ways you can compute $\boldsymbol{e} \in \mathbb{R}^N$ from $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and $\boldsymbol{s} \in \mathbb{R}^{d_2}$:

- Basic dot-product attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
  - Note: this assumes $d_1 = d_2$
  - This is the version we saw earlier
- Multiplicative attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
  - Where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix
- Additive attention: $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$
  - Where $\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $\boldsymbol{v} \in \mathbb{R}^{d_3}$ is a weight vector.
  - d_3 (the attention dimensionality) is a hyperparameter

Representations for a word

→ Word2Vec, GloVe, fastText

Pretrained word vectors

⇒ Start with random vectors & train them on our task of interest

→ But in most cases, use of pretrained word vectors helps, because we can train them for more words with much more data

Tips for unknown words

→ Simplest Solution

- Simplest and common solution:
- Train time: Vocab is {words occurring, say, ≥ 5 times} ∪ {<UNK>}
- Map **all** rarer (< 5) words to <UNK>, train a word vector for it
- Runtime: use <UNK> when out-of-vocabulary (OOV) words occur

- Problems:
  - No way to distinguish different UNK words, either for identity or meaning

- Solutions:
  1. Hey, we just learned about char-level models to build vectors! Let's do that!

Tips for unknown words with word vectors

- Especially in applications like question answering
  - Where it is important to match on word identity, even for words outside your word vector vocabulary
- 2. Try these tips (from Dhingra, Liu, Salakhutdinov, Cohen 2017)
  a. If the <UNK> word at test time appears in your unsupervised word embeddings, use that vector as is at test time.
  b. Additionally, for other words, just assign them a random vector, adding them to your vocabulary
- a. definitely helps a lot; b. may help a little more
- Another thing you can try:
  - Collapsing things to word classes (like unknown number, capitalized thing, etc. and having an <UNK-class> for each

Upto now we have 1 representation of words
↳ Word2Vec, GloVe, fasttext

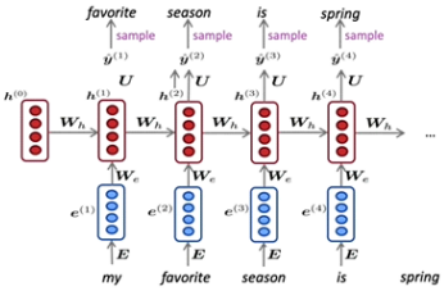
These have 2 problems

↳ One word may have multiple meanings
- Always the same representation for a **word type** regardless of the context in which a **word token** occurs
  - We might want very fine-grained word sense disambiguation

↳
- We just have one representation for a word, but words have different **aspects**, including semantics, syntactic behavior, and register/connotations

Did we all along have a solution to this problem?

- In, an NLM, we immediately stuck word vectors (perhaps only trained on the corpus) through LSTM layers
- Those LSTM layers are trained to predict the next word
- But those language models are producing context-specific word representations at each position!

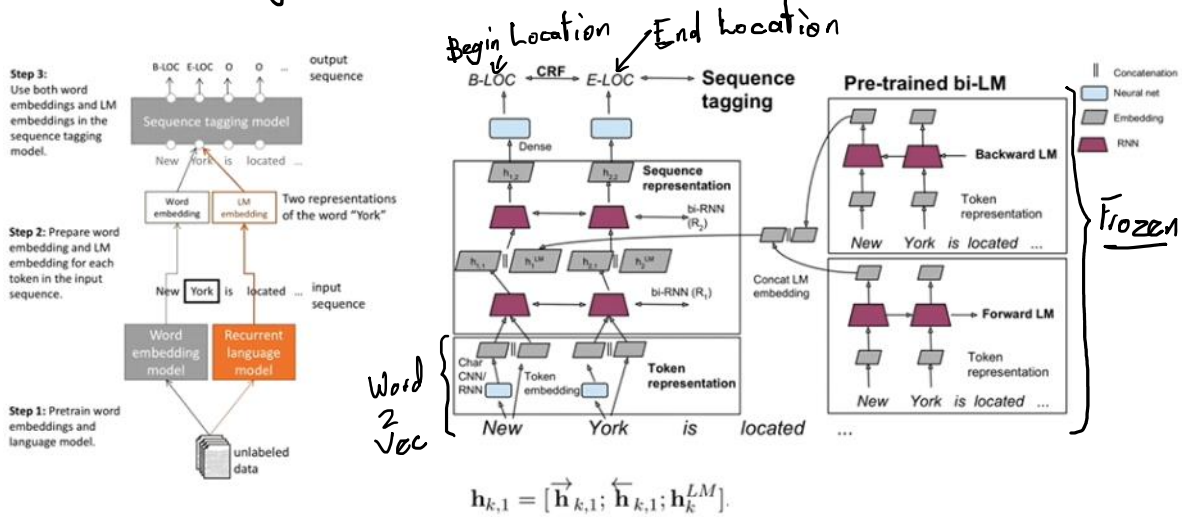


12

TagLM

→ Want meaning of word in context but standardly learn task RNN only on small task-labelled data (eg NER)

→ Why don't we do semi-supervised approach where we train NLM on large, unlabelled corpus rather than just word vectors?



$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}]$$

ELMo → Embeddings from language models

- Train a bidirectional LM
- Aim at performant but not overly large LM:
  - Use 2 biLSTM layers
  - Use character CNN to build initial word representation (only)
    - 2048 char n-gram filters and 2 highway layers, 512 dim projection
  - User 4096 dim hidden/cell LSTM states with 512 dim projections to next input
  - Use a residual connection
  - Tie parameters of token input and output (softmax) and tie

- Train a bidirectional LM
- Aim at performant but not overly large LM:
  - Use 2 biLSTM layers
  - Use character CNN to build initial word representation (only)
    - 2048 char n-gram filters and 2 highway layers, 512 dim projection
  - User 4096 dim hidden/cell LSTM states with 512 dim projections to next input
  - Use a residual connection
  - Tie parameters of token input and output (softmax) and tie these between forward and backward LMs

- ELMo learns task-specific combination of biLM representations
- This is an innovation that improves on just using top layer of LSTM stack

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

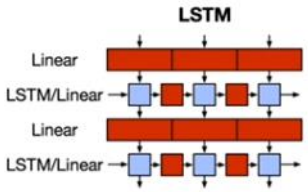
- γ^{task} scales overall usefulness of ELMo to task;
- $\mathbf{s}^{task}$ are softmax-normalized mixture model weights

ELMo results: Great for all tasks

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Motivation for Transformers

We want parallelization but RNNs are sequential

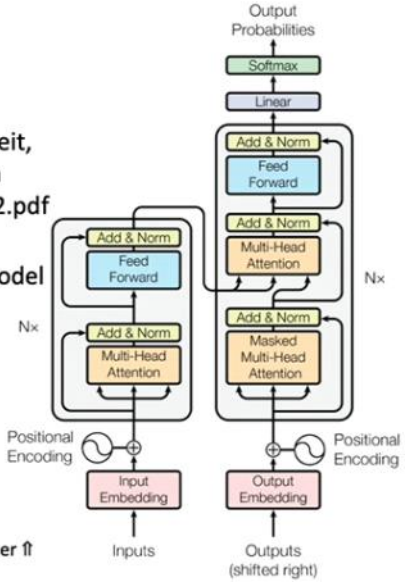


- Despite GRUs and LSTMs, RNNs still need attention mechanism to deal with long range dependencies – **path length** between states grows with sequence otherwise
- But if **attention** gives us access to any state… maybe we can just use attention and don't need the RNN? 🤔

Transformer Overview

Attention is all you need. 2017.
Aswani, Shazeer, Parmar, Uszkoreit,
Jones, Gomez, Kaiser, Polosukhin
https://arxiv.org/pdf/1706.03762.pdf

- Non-recurrent sequence-to-sequence encoder-decoder model
- Task: machine translation with parallel corpus
- Predict each translated word
- Final cost/error function is standard cross-entropy error on top of a softmax classifier



This and related figures from paper ⇑

38

Sasha Rush — Go through her tutorial on the topic

Dot-Product Attention (Extending our previous def.)

- Inputs: a query q and a set of key-value (k-v) pairs to an output
- Query, keys, values, and output are all vectors

- Output is weighted sum of values, where
- Weight of each value is computed by an inner product of query and corresponding key
- Queries and keys have same dimensionality d_k value have d_v

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

Dot-Product Attention – Matrix notation

- When we have multiple queries q, we stack them in a matrix Q:

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

- Becomes: $A(Q, K, V) = softmax(QK^T)V$

$[|Q| \times d_k] \times [d_k \times |K|] \times [|K| \times d_v]$

softmax row-wise

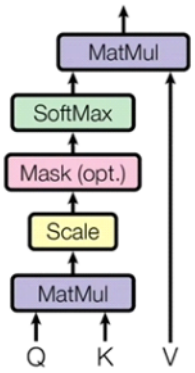
$= [|Q| \times d_v]$

Scaled Dot-Product Attention

- Problem: As d_k gets large, the variance of $q^T k$ increases → some values inside the softmax get large → the softmax gets very peaked → hence its gradient gets smaller.

- Solution: Scale by length of query/key vectors:

$$A(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



42

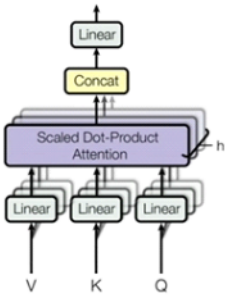
Self-attention in the encoder

- The input word vectors are the queries, keys and values
- In other words: the word vectors themselves select each other
- Word vector stack = Q = K = V
- We'll see in the decoder why we separate them in the definition

Multi-head attention

- Problem with simple self-attention:
- Only one way for words to interact with one-another
- Solution: Multi-head attention
- First map Q, K, V into h=8 many lower dimensional spaces via W matrices
- Then apply attention, then concatenate outputs and pipe through linear layer

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

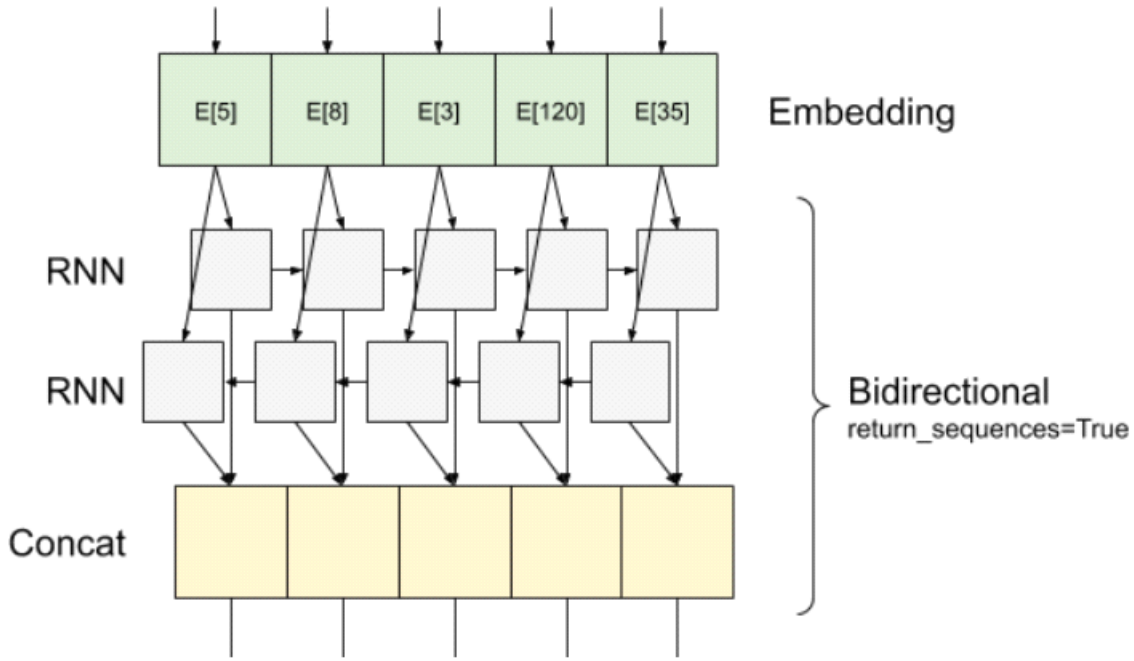


Implementation of Text Classification Using Tensorflow

13 October 2021 09:03

```
import numpy as np
import tensorflow_datasets as tfds
import tensorflow as tf
tfds.disable_progress_bar()

import matplotlib.pyplot as plt
def plot_graphs(history, metric):
	plt.plot(history.history[metric])
	plt.plot(history.history['val_'+metric],'')
	plt.xlabel("Epochs")
	plt.ylabel("metric")
	plt.legend([metric, 'val_'+metric])

dataset, info = tfds.load("imdb_reviews", with_info = True, as_supervised = True)
train_dataset, test_dataset = dataset["train"], dataset["test"]
train_dataset.element_spec

for example, label in train_dataset.take(1):
  print('text: ', example.numpy())
  print('label: ', label.numpy())

BUFFER_SIZE = 10000
BATCH_SIZE = 64

train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

for example, label in train_dataset.take(1):
	print('texts: ', example.numpy()[:3])
	print()
	print('labels: ', label.numpy()[:3])

VOCAB_SIZE = 10000
encoder = tf.keras.layers.experimental.preprocessing.TextVectorization(max_tokens = VOCAB_SIZE)
encoder.adapt(train_datasets.map(lamda text, label: text))

vocab = np.array(encoder.get_vocabulary())
vocab[:20]

encoded_example = encoder(example)[:3].numpy()
encoded_example

for n in range(3):
  print("Original: ", example[n].numpy())
  print("Round-trip: ", " ".join(vocab[encoded_example[n]]))
  print()

model = tf.keras.Sequential([encoder, tf.keras.layers.Embedding(input_dim =
len(encoder.get_vocabulary()), output_dim = 64, mask_zero = True),
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)), tf.keras.layers.Dense(64, activation = 'relu'),
tf.keras.layers.Dense(1)])

print([layer.supports_masking for layer in model.layers])
```

```
# predict on a sample text without padding.

sample_text = ('The movie was cool. The animation and the graphics '
               'were out of this world. I would recommend this movie.')
predictions = model.predict(np.array([sample_text]))
print(predictions[0])

# predict on a sample text with padding

padding = "the " * 2000
predictions = model.predict(np.array([sample_text, padding]))
print(predictions[0])

model.compile(loss = tf.keras.losses.BinaryCrossentropy(from_logits = True), optimizer =
tf.keras.optimizers.Adam(1e-4), metrics = ['accuracy'])

history = model.fit(train_dataset, epochs = 10, validation_data = test_dataset,  validation_steps = 30)

test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)

plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
plot_graphs(history, 'accuracy')
plt.ylim(None, 1)
plt.subplot(1, 2, 2)
plot_graphs(history, 'loss')
plt.ylim(0, None)

sample_text = ('The movie was cool. The animation and the graphics '
               'were out of this world. I would recommend this movie.')
predictions = model.predict(np.array([sample_text]))
```



```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 64, mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1)
])
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)

# predict on a sample text without padding.

sample_text = ('The movie was not good. The animation and the graphics '
               'were terrible. I would not recommend this movie.')
predictions = model.predict(np.array([sample_text]))
print(predictions)

plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plot_graphs(history, 'accuracy')
plt.subplot(1, 2, 2)
plot_graphs(history, 'loss')
```

Text Generation with RNNs on Tensorflow

13 October 2021 10:34

```
import tensorflow as tf
from tensorflow.keras.layers.experimental import preprocessing

import numpy as np
import os
import time

path_to_file = tf.keras.utils.get_file('shakespeare.txt',
'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt')

text = open(path_to_file, 'rb').read().decode(encoding = 'utf-8')
print(f'Length of text: {len(text)} characters')

print(text[:250])
vocab = sorted(set(text))
print(f'{len(vocab)} unique characters')

example_texts = ['abcdefg', 'xyz']
chars = tf.string.unicode_split(example_texts, input_encoding = 'UTF-8')
print(chars)

ids_from_chars = preprocessing.StringLookup(vocabulary = list(vocab), mask_token = None)
ids = ids_from_chars(chars)
print(ids)

chars_from_ids = preprocessing.StringLookup(vocabulary = ids_from_chars.get_vocabulary, invert =
True, mask_token = None)

chars = chars_from_ids(ids)
print(chars)

def text_from_ids(ids):
    return tf.strings.reduce_join(chars_from_ids(ids), axis = -1)

all_ids = ids_from_chars(tf.strings.unicode_split(text, 'UTF-8'))
print(all_ids)

ids_dataset = tf.data.Dataset.from_tensor_slices(all_ids)

for ids in ids_dataset.take(1):
    print(chars_from_ids(ids).numpy().decode('UTF-8')

seq_length = 100
examples_per_epoch = len(text)//(seq_length +1)

sequences = ids_dataset.batch(seq_length+1, drop_remainder = True)
for seq in sequences.take(1):
    print(chars_from_ids(seq))

for seq in sequences.take(1):
    print(text_from_ids(seq).numpy())

def split_input_target(sequence):
```

```
    input_text = sequence[0:-1]
    target_text = sequence[1:]
    return input_text, target_text

split_input_target(list('Tensorflow'))

dataset = sequences.map(split_input_target)

for input_example, target_example in dataset.take(1):
    print("Input: ", text_from_ids(input_example).numpy())
    print("Target: ", text_from_ids(target_example).numpy())

BATCH_SIZE = 64
BUFFER_SIZE = 10000

dataset = (dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder =
True).prefetch(tf.data.experimental.AUTOTUNE))

print(dataset)

vocab_size = vocab
embedding_dim = 256
rnn_units = 1024

class myModel(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, rnn_units):
        super().__init__(self)
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(rnn_units, return_sequences = True, return_state = True)
        self.dense = tf.keras.layers.Dense(vocab_size)
    def call(self, inputs, states = None, return_state = False, training = False):
        x = inputs
        x = self.embedding(x, training = training)
        if states is None:
            states = self.gru.get_initial_state(x)
        x, states = self.gru(x, initial_state = states, training = training)
        x = self.dense(x, training = training)
        if return_state:
            return x, states
        else:
            return x

model = myModel(vocab_size = len(ids_from_chars.get_vocabulary()), embedding_dim =
embedding_dim, rnn_units = rnn_units)

model.summary()

for input_example_batch, target_example_batch in dataset.take(1):
    example_batch_predictions = model(input_example_batch)
    print(example_batch_predictions.shape, "# (batch_size, sequence_length, vocab_size)")


sampled_indices = tf.random.categorical(example_batch_predictions[0], num_samples=1)
sampled_indices = tf.squeeze(sampled_indices, axis=-1).numpy()

sampled_indices
```

```
print("Input:\n", text_from_ids(input_example_batch[0]).numpy())
print()
print("Next Char Predictions:\n", text_from_ids(sampled_indices).numpy())

loss = tf.losses.SparseCategoricalCrossentropy(from_logits = True)
example_batch_loss = loss(target_example_batch, example_batch_predictions)
mean_loss = example_batch_loss.numpy().mean()

print("Prediction shape: ", example_batch_predictions.shape, " # (batch_size, sequence_length,
vocab_size)")
print("Mean loss:        ", mean_loss)
model.compile(optimizer = 'adam', loss = loss)
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, 'ckpt_{epoch}')

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath = checkpoint_prefix,
save_weights_only = True)

EPOCHS = 20
history = model.fit(dataset, epochs = EPOCHS, callbacks = [checkpoint_callback])

class OneStep(tf.keras.Model):
    def __init__(self, model, chars_from_ids, ids_from_chars, temperature = 1.0):
        super().__init__()
        self.temperature = temperature
        self.model = model
        self.chars_from_ids = chars_from_ids
        self.ids_from_chars = ids_from_chars

        # Create a mask to prevent [UNK] from being generated
        skip_ids = self.ids_from_chars(['[UNK]'])[:, None]

        sparse_mask = tf.SparseTensor(values = [-float(inf)] * len(skip_ids), indices = skip_ids,
        dense_shape = [len(ids_from_chars.get_vocabulary())])
        self.prediction_mask = tf.sparse.to_dense(sparse_mask)

    @tf.function
    def generate_one_step(self, inputs, states = None):
        input_chars = tf.strings.unicode_split(inputs, 'UTF-8')
        input_ids = self.ids_from_chars(input_chars).to_tensor()

        # Run the model.
        # predicted_logits.shape is [batch, char, next_char_logits]
        predicted_logits, states = self.model(inputs=input_ids, states=states,
                                              return_state=True)
        # Only use the last prediction.
        predicted_logits = predicted_logits[:, -1, :]
        predicted_logits = predicted_logits/self.temperature
        # Apply the prediction mask: prevent "[UNK]" from being generated.
        predicted_logits = predicted_logits + self.prediction_mask

        # Sample the output logits to generate token IDs.
        predicted_ids = tf.random.categorical(predicted_logits, num_samples=1)
        predicted_ids = tf.squeeze(predicted_ids, axis=-1)

        # Convert from token ids to characters
        predicted_chars = self.chars_from_ids(predicted_ids)
```

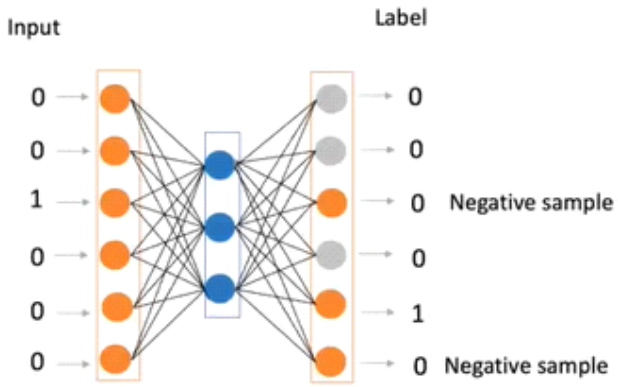
```
            # Return the characters and model state.
            return predicted_chars, states

one_step_model = OneStep(model, chars_from_ids, ids_from_chars)

start = time.time()
states = None
next_char = tf.constant(['ROMEO:'])
result = [next_char]

for n in range(1000):
  next_char, states = one_step_model.generate_one_step(next_char, states=states)
  result.append(next_char)

result = tf.strings.join(result)
end = time.time()
print(result[0].numpy().decode('utf-8'), '\n\n' + '_'*80)
print('\nRun time:', end - start)
```

Word2Vec Implementation Using Tensorflow

01 November 2021 11:13

```
!pip install -q tqdm

import tensorflow as tf
print(tf.__version__)

import io
import itertools
import numpy as np
import os
import re
import string
import tqdm
from tensorflow.keras import Model, Sequential
from tensorflow.keras.layers import Activation, Dense, Dot, Embedding, Flatten,
GlobalAveragePooling1D, Reshape
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization


# Data Preparation
# 1. Negative Sampling
# 2. Subsampling

path_to_file = tf.keras.utils.get_file('shakespeare.txt', 'https://.....')

with open(path_to_file) as f:
        lines = f.read().splitlines()
for line in lines[:20]:
        print(line)

# Load data with tf.data.Dataset API
text_ds = tf.data.TextLineDataset(path_to_file).filter(lambda x: tf.cast(tf.strings.length(x), bool))

type(text_ds)

sentences = list(text_ds.as_numpy_iterator())
print(len(sentences))

for stc in sentences[:5]:
        print(stc)

def custom_standardization(input_data):
        lowercase = tf.strings.lower(input_data)
        return tf.strings.regex_replace(lowercase, '[%s]' % re.escape(string.punctuation), '')

vocab_size = 4096
sequence_length = 10
vectorize_layer = TextVectorization(standardize = custom_standardization, max_tokens =
vocab_size, output_mode = 'int', output_sequence_length = sequence_length)
vectorize_layer.adapt(text_ds,batch(1024))
inverse_vocab = vectorize_layer.get_vocabulary()
print(len(inverse_vocab))
```

```
print(inverse_vocab[:20])

text_vector_ds = text_ds.batch(1024).prefetch(tf.data.AUTOTUNE).map(vectorize_layer).unbatch()

sequences = list(text_batch_ds.as_numpy_iterator())
print(len(sequences))

for seq in sequences[:5]:
        print(f'{seq} => {[inverse_vocab[i]] for i in seq}')

vocab = {}
for index, token in enumerate(inverse_vocab):
        vocab[token] = index

print(len(vocab))

sentence = 'Very well and could be content to give him good report fort'
tokens = list(sentence.lower().split())
example_sequence = [vocab[word] for word in tokens]
print(example_sequence)

window_size = 2
positive_skip_grams, _ = tf.keras.preprocessing.sequence.skipgrams(example_sequence,
vocabulary_size = vocab_size, window_size = window_size, negative_sampling = 0)

print(len(positive_skip_grams))
for target, context in positive_skip_grams[:5]:
        print(f"({target},{context}): ({inverse_vocab[target]}, {inverse_vocab[context]})")

# Negative Sampling
```




```
# Negative sampling for one skip-gram
# Get target and context words for one positive skip-gram
target_word, context = positive_skip_gram[1]

# Set the number of negative samples per positive context
num_ns = 4
SEED = 35
context_class = tf.reshape(tf.constant(context_word, dtype = 'int64'), (1,1))
negative_sampling_candidates, _ = tf.random.log_uniform_candidate_sampler(true_classes =
context_class, num_true = 1, num_sampled = num_ns, unique = True, range_max = vocab_size, seed
= SEED, name = 'negative_sampling')

print(negative_sampling_candidates)
print([inverse_vocab[index.numpy()] for index in negative_sampling_candidates])
 negative_sampling_candidates = tf.expand_dims(negative_sampling_candidates, 1)
```

```
context = tf.concat([context_class, negative_sampling_candidates], 0)
label = tf.constant([1] + [0]*num_ns, dtype = 'int64')
target = tf.squeeze(target_word)
context = tf.squeeze(context)
label = tf.squeeze(label)

print(f"target_index          :  {target}")
print(f"target_word           : {inverse_word[target_word]}")
print(f"context_indices       : {context}")
print(f"context_words         : {inverse_vocab[c.numpy()] for c in context ]}")
print(f"label                 : {label}")

# Compile all steps into one functions

def generate_training_data(sequences, window_size, num_ns, vocab_size, seed):
        targets_contexts, labels = [], [], []
        sampling_table = tf.keras.preprocessing.sequence.make_sampling_table(vocab_size)
        for sequence in tqdm.tqdm(sequences):
                positive_skip_grams, _ = tf.keras.preprocessing.sequence.skipgrams(sequence,
                vocabulary_size = vocab_size, sampling_table = sampling_table, window_size =
                window_size, negative_samples = 0)
                for target_word, context_word in positive_skip_grams;
                        context_class = tf.expand_dims(tf.constant([context_word], dtype = 'int64'), 1)
                        negative_sampling_candidates, _ =
                        tf.random.log_uniform_candidate_sampler(true_classes = context_class,
                        num_true = 1, num_sampled = num_ns, unique = True, range_max = vocab_size,
                        seed = SEED, name = 'negative_sampling')
                        negative_sampling_candidates = tf.expand_dims(negative_sampling_candidates,
                        1)
                        context = tf.concat([context_class, negative_sampling_candidates], 0)
                        label = tf.constant([1] + [0]*num_ns, dtype = 'int64')
                        targets.append(target_word)
                        contexts.append(context)
                        labels.append(label)
        return targets, contexts, labels

sampling_table = tf.keras.preprocessing.sequence.make_sampling_table(size = 10)
print(sampling_table)

vocab_size

targets, contexts, labels = generate_training_data(sequences = sequences, window_size = 2, num_ns
= 4, vocab_size = vocab_size, seed = SEED)
print(len(targets), len(contexts), len(labels))

print("target, context, label")
for target, context, label in zip(targets[:5], contexts[:5], labels[:5]):
        print(target, context, label)

# Subclassed Word2Vec Model
num_ns = 4
class Word2Vec(Model):
        def __init__(self, vocab_size, embedding_dim):
                super(Word2Vec, self).__init__()
                self.target_embedding = Embedding(vocab_size, embedding_dim, input_length = 1,
                name = "w2v_embedding")
                self.context_embedding = Embedding(vocab_size, embedding_dim, input_length =
```

```
            num_ns + 1)
            self.dots = Dot(axes = (3,2))
            self.flatten = Flatten()
        def call(self, pair):
            target, context = pair
            we = self.target_embedding(target)
            ce = self.context_embedding(context)
            dots = self.dots([ce, we])
            return self.flatten(dots)

embedding_dim = 128
word2vec = Word2Vec(vocab_size, embedding_dim)
word2vec.compile(optimizer = 'adam', loss = tf.keras.losses.CategoricalCrossentropy(from_logits =
True))

BATCH_SIZE = 1024
BUFFER_SIZE = 10000
dataset = tf.data.Dataset.from_tensor_slices((targets, contexts), labels)
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder = True)
print(dataset)

dataset = dataset.cache().prefetch(tf.data.AUTOTUNE)
print(dataset)

tensorboard_callback = tf.keras.callbacks.Tensorboard(log_dir = 'logs')

word2vec.fit(dataset, epochs = 20, callbacks = [tensorboard_callback])
%tensorboard --logdir '{path_to_your_logs}'

weights = word2vec.get_layer('w2v_embedding').get_weights()[0]
vocab = vectorize_layer.get_vocabulary()

out_v = io.open('vectors.tsv', 'w', encoding = 'utf-8')
out_m = io.open('metadata.tsv', 'w', encoding = 'utf-8')

for index, word in enumerate(vocab):
    if index == 0: continue
    vec = weights[index]
    out_v.write('\t'.join([str(x) for x in vec] + '\n'))
    out_m.write(word + '\n')

out_v.close()
out_m.close()
```

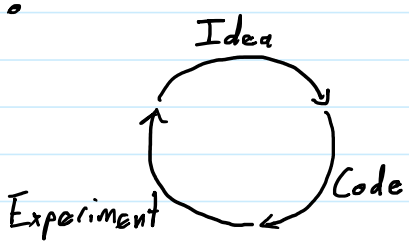
Setting up your Machine Learning Application

20 October 2021 11:12

Train/Dev/Test Set

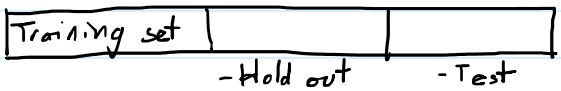
Applied ML is a highly iterative process

layers
hidden units
learning rates
activation functions
...



NLP, Vision, Speech, Structural Data
↳ Ads, Search (results), Security, Logistics

Intuitions often do not transfer well between domains

Data 

	Train	Test	Train	Dev	Test
Previous era	70	30	60	20	20

Big data : 10 M examples
↳ Just big enough to evaluate performance

98/1/1%

99.5/0.4/0.1

Mismatched train/test distribution

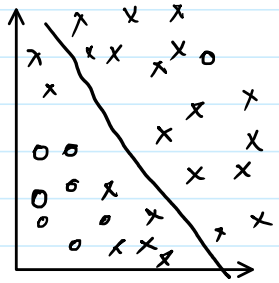
Training set: Dev / Test sets:

Cat pictures from webpages — Cat pictures from users using your app

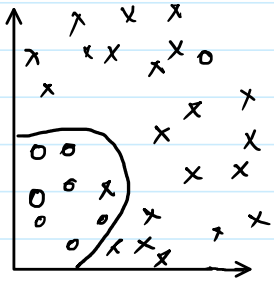
→ Make sure dev & test set come from the same distribution

→ Not having a test set might be okay (Only dev set)

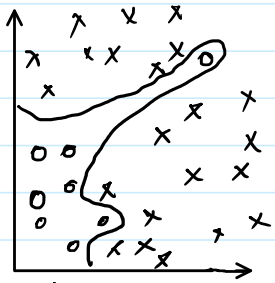
Bias/Variance



high bias



just right



high variance

Bias & Variance

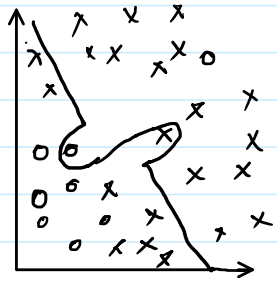
Cat classification

Training set error : 1%	15%	15%	0.5%
Dev set error : 11%	16%	30%	1%
high variance	high bias	high bias & high variance	low bias & low variance

Humans ≈ 0%

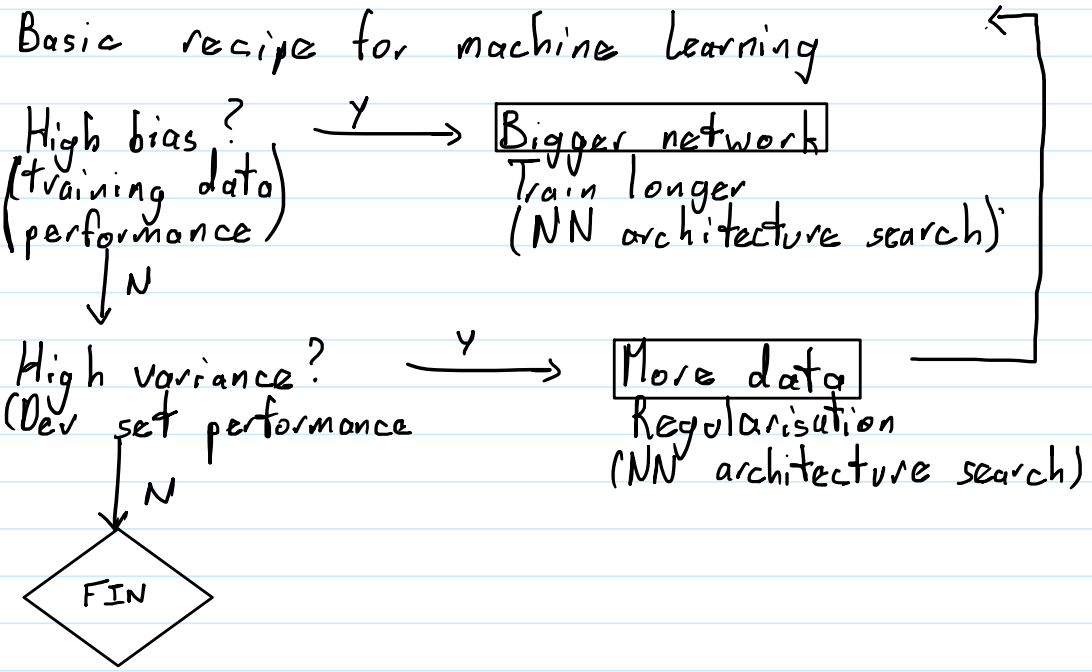
Optimal (Bayes) error : 0%

High bias & High Variance



Basic recipe for machine learning ↰

Basic recipe for machine learning



Regularization

Logistic Regression

$\min_{w,b} J(w,b)$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2$$

regularisation parameter

L2 regularisation $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j = w^T w$

L1 regularisation $\frac{\lambda}{2m}\sum_{i=1}^{n_x}|w| = \frac{\lambda}{2m}\|w\|_1$

$$J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{n} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|_F^2$$

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(w_{ij}^{[l]})^2$$

Frobenius Norm

$$dW^{[l]} = (\text{from backprop}) + \frac{\lambda}{m}W^{[l]}$$

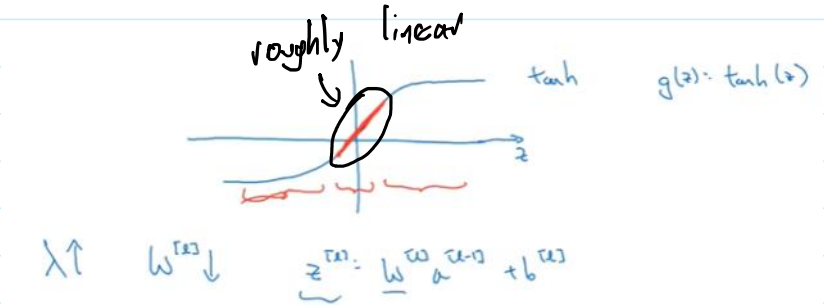
$$W^{[l]} = W^{[l]} - \alpha\, dW^{[l]}$$

$$W^{[l]} = W^{[l]} - \alpha\, dW^{[l]} \longleftarrow$$

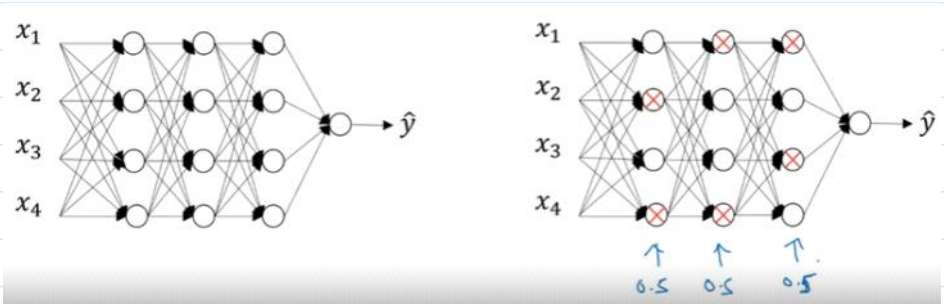
$$W^{[l]} = \underbrace{\left(1 - \frac{\alpha\lambda}{m}\right)}_{\text{Weight decay}} W^{[l]} - \alpha \text{ (from backprop)}$$

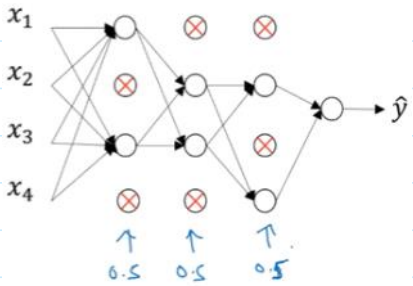
Why regularisation prevents overfitting?

$$J(w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|W^{[L]}\|_F^2$$



Dropout Regularisation





Implementing Dropout ("Inverted Dropout")

Illustrate with layer $l = 3$; keep_prob = 0.8

```
d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep_prob
a3 = np.multiply(a3, d3)
a3 /= keep_prob
```

$z^{[4]} = W^{[4]} a^{[3]} + b^{[4]}$

$a^{[3]}$ ← reduced by 20% ↓ $/= 0.8$

Making predictions at test time

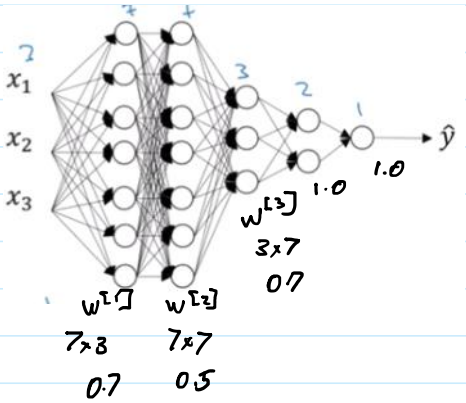
$a^{[0]} = X$

No dropout

Why does dropout work?

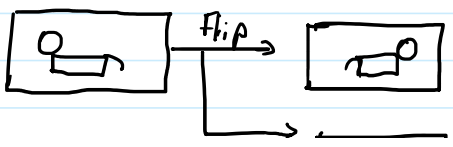
Intuition: Can't rely on any one feature so have to spread out weights

Similar effect to L2 (only that regularisation weights of different units are different)

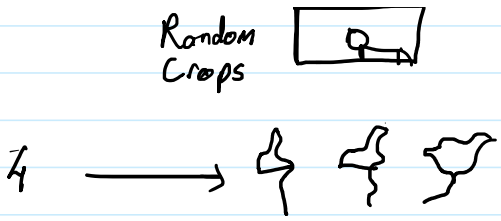


Other regularisation techniques

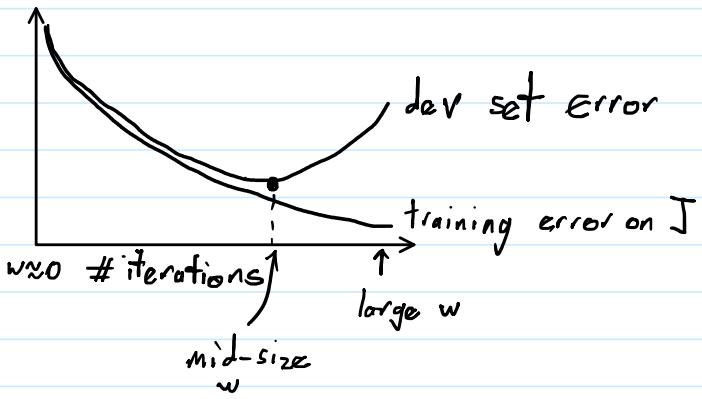
↳ Data augmentation



Random Crops



Early Stopping



Downside (Orthogonalisation)
- Optimize cost function J
  - Gradient Descent
- Don't overfit
  - Regularisation, . . .

Early Stopping combines these two tasks because
→ stopping early does not optimize cost fn
→ Simultaneously tries not to overfit

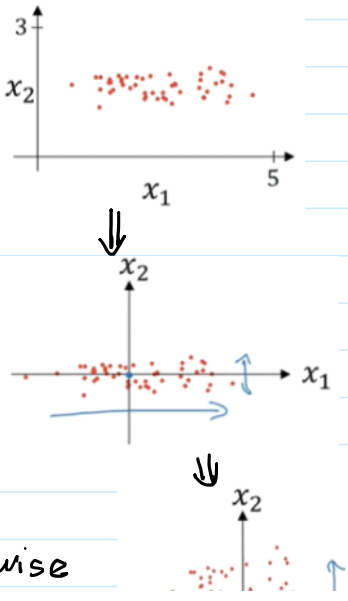
No longer can you have independent control of these tasks

Normalising inputs

Subtract mean

$$\mu = \frac{1}{m} \sum_{i=1}^{m} X^{(i)}$$

$$X := X - \mu$$

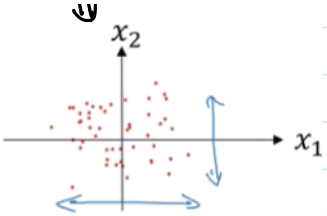


Divide by Variance

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} X^{(i)} **2$$

↖ elementwise

$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} **2$ ← elementwise

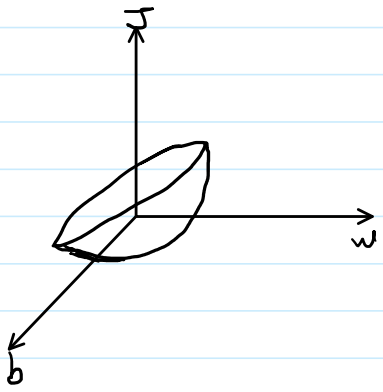


Note: Use same μ & σ from training set on test set & dev set

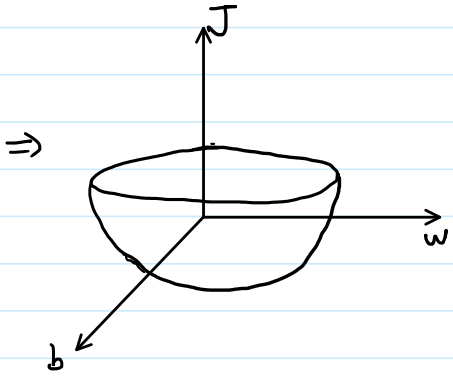
Why normalize inputs?

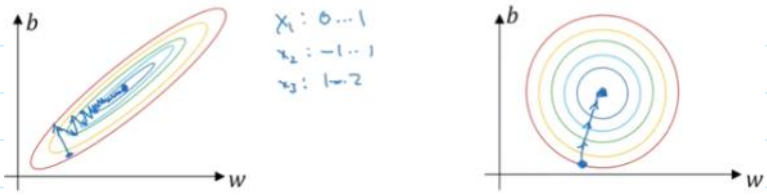
$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

Unnormalized

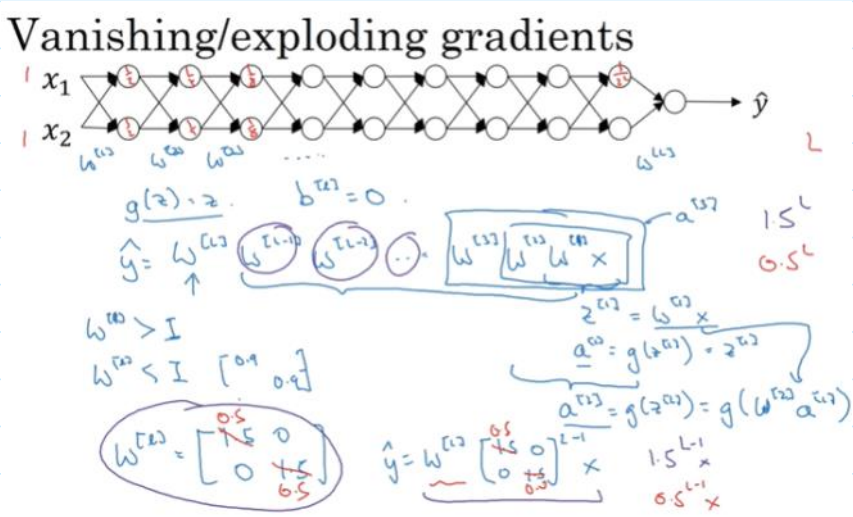


Normalized

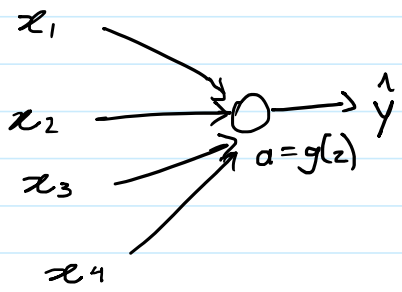




Vanishing & Exploding Gradients



Weight initialization for deep neural networks



$z = w_1 x + w_2 x_2 + \ldots w_n x_n$

Large n $\longrightarrow$ smaller w_i

$Var(w_i) = \frac{1}{n} = \frac{2}{n}$

$W^{[l]} = np.random.randn(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$

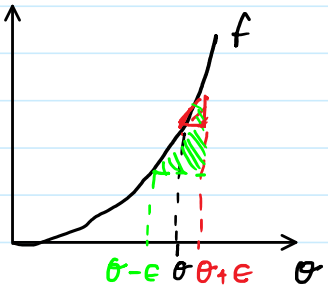
$g^{[l]}(z) = ReLU(z)$

Other variants

tanh : $\sqrt{\frac{1}{n^{[l-1]}}}$

Xavier Initialisation $\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$

Numerical Approximation of Gradients



Checking your derivative computation

$$\frac{df}{d\theta} = \frac{f(\theta+\epsilon) - f(\theta-\epsilon)}{2\epsilon}$$

Mini-batch gradient descent

20 October 2021 16:08

Vectorization allows you to efficiently compute on m examples

$X_{(n_x,m)} = [x^{(1)}, x^{(2)} \ldots\ldots x^{(m)}]$

$Y_{(1,m)} = [y^{(1)}, y^{(2)} \ldots\ldots y^{(m)}]$

What if $m = 5,000,000$?

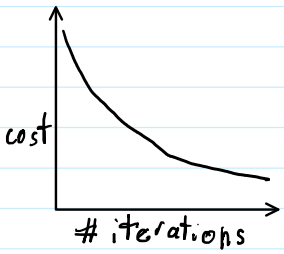
↳ mini-batches of 1000 each

$X = [\underbrace{x^{(1)}, x^{(2)} \ldots x^{(1000)}}_{X^{\{1\}}} | \underbrace{x^{(1001)} \ldots x^{(2000)}}_{X^{\{2\}}} | \ldots | \underbrace{\ldots x^{(m)}}_{X^{\{5000\}}}]$

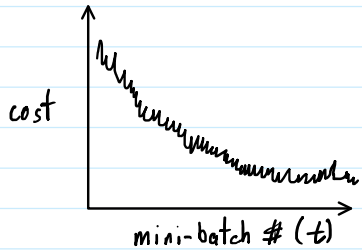
$Y = [\underbrace{y^{(1)}, y^{(2)} \ldots y^{(1000)}}_{Y^{\{1\}}} | \underbrace{y^{(1001)} \ldots y^{(2000)}}_{Y^{\{2\}}} | \ldots | \underbrace{\ldots y^{(m)}}_{Y^{\{5000\}}}]$

Understanding mini-batch gradient descent

Batch gradient descent



Mini batch gradient descent

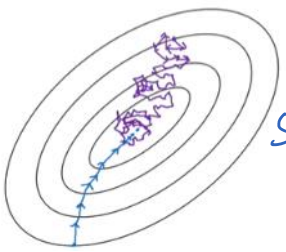


Choosing your minibatch size

If minibatch size = m : Batch gradient descent $(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$

If minibatch size = 1 : Stochastic gradient descent — Every example is its own

In practice : Somewhere b/w 1 & m



Stochastic Gradient Descent

↓

Lose speedup from vectorisation

In between

↓

Fastest learning

- Vectorization
- Make progress without processing the entire training set

Batch gradient descent

↓

Too long per iteration

Choosing mini-batch size

If small training set: Use batch gradient descent ($m \leq 2000$)

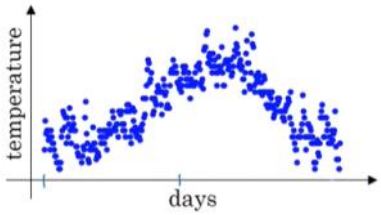
Typical mini batch sizes:

64, 128, 256, 512

Make that mini batch fits in CPU/GPU memory

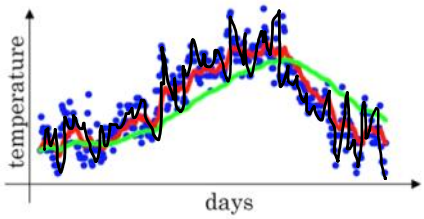
Exponentially Weighted Averages

$\theta_1 = 40°F$ 4°C
$\theta_2 = 49°F$ 9°C
$\theta_3 = 45°F$ ⋮
⋮
$\theta_{180} = 60°F$ 15°C
$\theta_{181} = 56°F$ ⋮
⋮



$V_0 = 0$
$V_1 = 0.9 V_0 + 0.1 \theta_1$
$V_2 = 0.9 V_1 + 0.1 \theta_2$
⋮
$V_t = 0.9 V_{t-1} + 0.1 \theta_t$

$V_t = \beta V_{t-1} + (1-\beta)\theta_t$



We can think of V_t as averaging over $\frac{1}{1-\beta}$ days temperature

$\beta = 0.9 \sim 10$ days
$\beta = 0.98 \sim 50$ days
$\beta = 0.5 \sim 2$ days

Understanding Exponentially weighted averages

$V_t = \beta V_{t-1} + (1-\beta)\theta_t$

$V_{100} = 0.9 V_{99} + 0.1 \theta_{100}$
$V_{99} = 0.9 V_{98} + 0.1 \theta_{99}$
$V_{98} = 0.9 V_{97} + 0.1 \theta_{98}$
...

$V_{100} = 0.1\theta_{100} + 0.9(0.1\theta_{99} + 0.9(0.1\theta_{98} + 0.9$

$V_{100} = 0.1[\theta_{100} + 0.9\theta_{99} + 0.9^2\theta_{98} + \ldots 0.9^{99}\theta_1]$

$(1-\epsilon)^{1/\epsilon} = \frac{1}{e}$

Implementing exponentially weighted averages

$v_\theta = 0$
$v_\theta = \beta v + (1-\beta)\theta_1$
$v_\theta = \beta v + (1-\beta)\theta_2$

$v_\theta = 0$
Repeat {
  Get next θ_t
  $v_\theta = \beta v_\theta + (1-\beta)\theta_t$
}

Bias correction in exponentially weighted averages

$$\frac{v_t}{1-\beta^t}$$

$t=2 : 1-\beta^t = 1-0.98^2 = 0.0396$

$$\frac{v_2}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$$

As $t \uparrow$, $\beta^t \downarrow$, $1-\beta^t \to 1$ & point bias gets smaller

Gradient descent with momentum

Momentum
On iteration t:
Compute dW, db on current minibatch
$v_{dW} = \beta v_{dW} + (1-\beta)dW$
$v_{db} = \beta v_{db} + (1-\beta)db$
friction → velocity ← acceleration
$W = W - \alpha v_{dW}$
$b = b - \alpha v_{db}$

Hyperparameters: α, β $\beta = 0.9$

Generally bias correction is not needed

$v_{dW} = 0$
$v_{db} = 0$

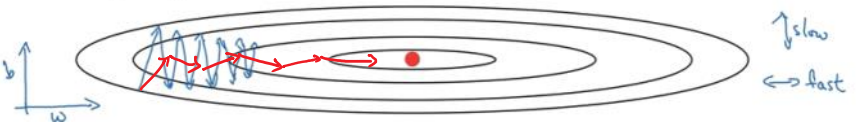
Sometimes :-

$V_{dw} = \beta V_{dw} + dW$
$V_{db} = \beta V_{db} + db$

α needs to be scaled appropriated to compensate for the absence of $(1-\beta)$ before dW

RMS Prop

RMSprop



On iteration t:

Compute dW, db on current minibatch

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dW^2$ ← element-wise ← small
$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$ ← large

$$w = w - \alpha \frac{dW}{\sqrt{S_{dw}+\epsilon}} \; ; \; b = b - \alpha \frac{db}{\sqrt{S_{db}+\epsilon}}$$

$\epsilon = 10^{-8}$

Optimization Algorithm

Adam optimization algorithm

$V_{dw} = 0, \; S_{dw} = 0$
$V_{db} = 0, \; S_{db} = 0$

On iteration t:
Compute dW, db using current minibatch
$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dW$
$V_{db} = \beta_1 V_{db} + (1-\beta_1) db$ } momentum β_1
$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dW^2$
$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$ } RMS prop β_2

$$V_{dw}^{corrected} = \frac{V_{dw}}{1-\beta^t}$$
$$V_{db}^{corrected} = \frac{V_{db}}{1-\beta_1^t}$$

$$S_{dw}^{corrected} = \frac{S_{dw}}{1-\beta_2^t}$$
$$S_{db}^{corrected} = S_{db}$$

$$S_{dw}^{corrected} = \frac{S_{dw}}{1-\beta_2^t}$$

$$S_{db}^{corrected} = \frac{S_{db}}{1-\beta_2^t}$$

$$w = w - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \epsilon}$$

$$b = b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \epsilon}$$

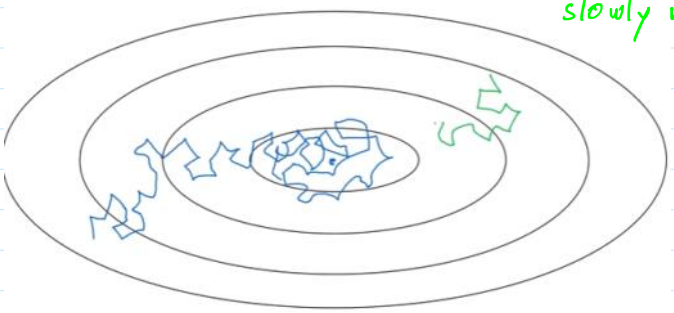
Hyperparameters choice

α : needs to be tuned
β_1 : 0.9 (dW)
β_2 : 0.999 (dW^2)
ϵ : 10^{-8}

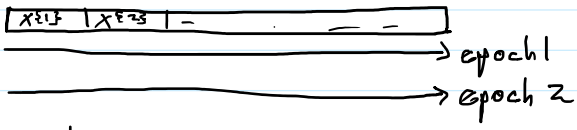
Adam ⇒ Adaptive Moment estimation

Learning Rate Decay

slowly reduce α



1 epoch ⇒ 1 pass through data



$$\alpha = \frac{1}{1 + \text{decay rate} * \text{epoch num}} \alpha_0$$

↳ another hyperparameter

Epoch	α
1	0.1
2	0.067
3	0.05
4	0.04
⋮	⋮

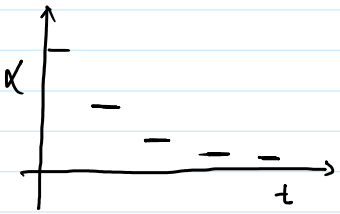
$\alpha_0 = 0.2$

decay rate = 1

Other learning rate decay methods

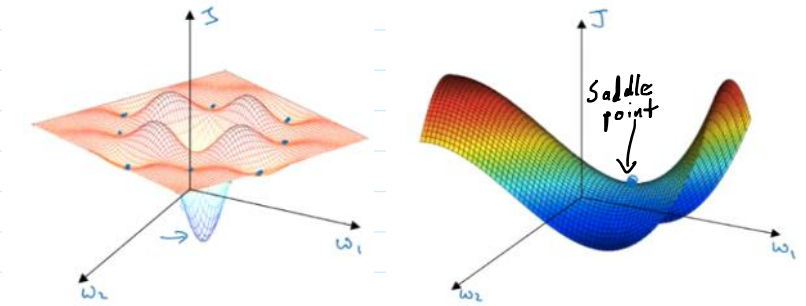
$\alpha = 0.95^{\text{epoch num}} \cdot \alpha_0$ — exponential decay

$\alpha = \frac{k}{\sqrt{\text{epoch num}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$

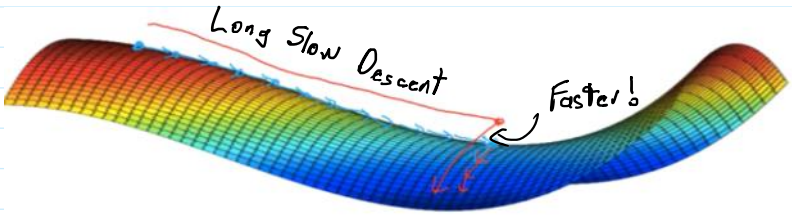


discrete staircase

The problem of local optima



Problem of plateaus



→ Unlikely to get stuck in bad local optima
→ Plateaus can make learning slow

Hyperparameter Tuning, Batch Normalization and Programming Frameworks

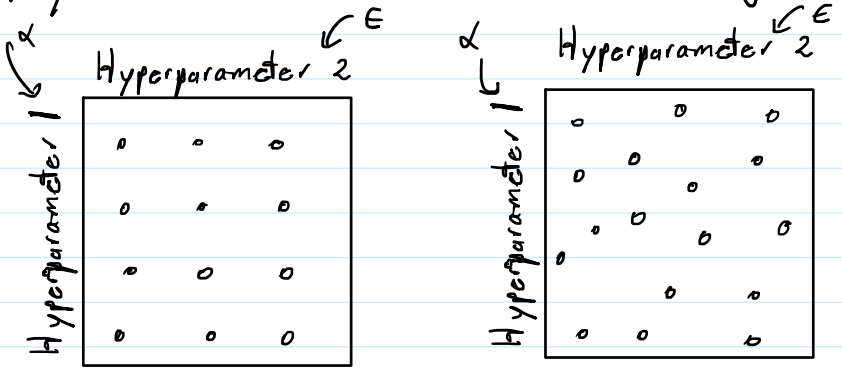
21 October 2021 13:10

Tuning Process

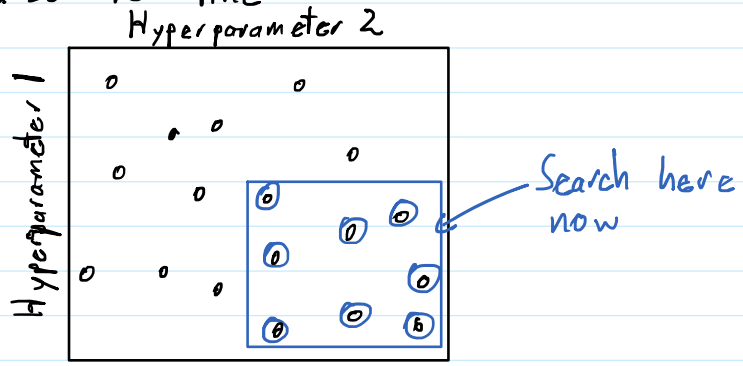
- α → most important
- α → learning rate
- β → momentum → 0.9
- $\beta_1, \beta_2, \epsilon$ → Adam ($0.9, 0.999, 10^{-9}$)
- # layers
- # hidden units
- learning rate decay
- minibatch size

2nd most important

Try random values : Don't use a grid!



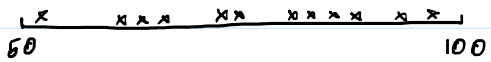
Coarse to fine



Using an appropriate scale to pick hyperparameters

Picking hyperparameters at random

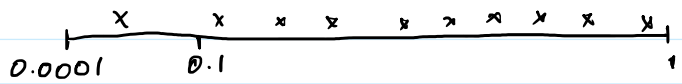
$n^{[l]} = 50, \ldots, 100$



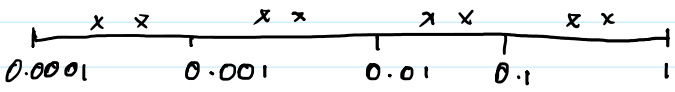
layers L : 2-4

2, 5, 4

Appropriate scale for hyperparameters



Instead use the log scale



$r = -4 * np.random.rand()$

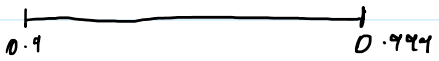
$\alpha = 10^r$

In general for sample b/w 10^a & 10^b

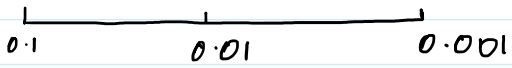
$r \in [a, b]$

$\alpha = 10^r$

$\beta = 0.9 \quad \ldots \quad 0.999$



$1-\beta = 0.1 \quad \ldots \quad 0.001$



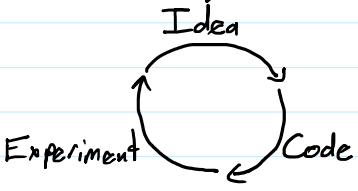
$r \in [-3, -1]$

$1-\beta = 10^r$

$\beta = 1-10^r$

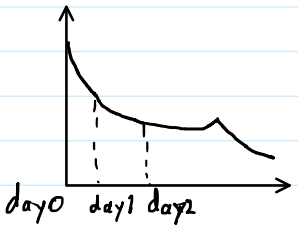
Hyperparameter Tuning in Practice

Re-test hyperparameters occasionally



- NLP, Vision, Speech, Ads, logistics
- Intvitions do get stale Re-evaluate occassionally

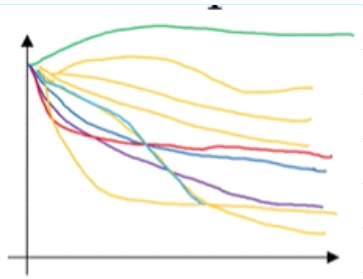
Babysitting one model



- What happens if you don't have computational capacity

Pandas approach (Limited computation resources

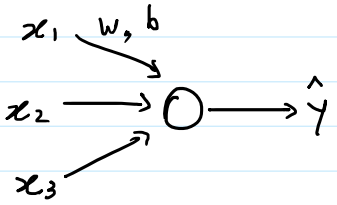
Training many models in parallel



Caviar approach
(If lot of computational resources)

Normalizing Activations in a Network

Batch Normalisation

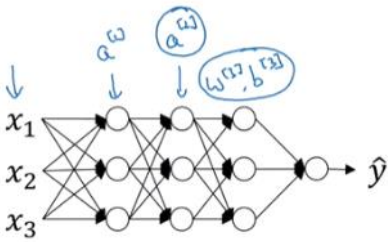


$$\mu = \frac{1}{m}\sum_i x^{(i)}$$

$$X = X - \mu$$

$$\sigma^2 = \frac{1}{m}\sum_i x^{(i)2}$$

$$X = \frac{X}{\sigma}$$



Can we normalise $a^{[2]}$ so as to train $w^{[3]}, b^{[3]}$ faster?

Normalise $z^{[2]}$

Given some intermediate values in NN

$z^{(1)}, \ldots, z^{(m)}$

$z^{[l](i)}$

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$

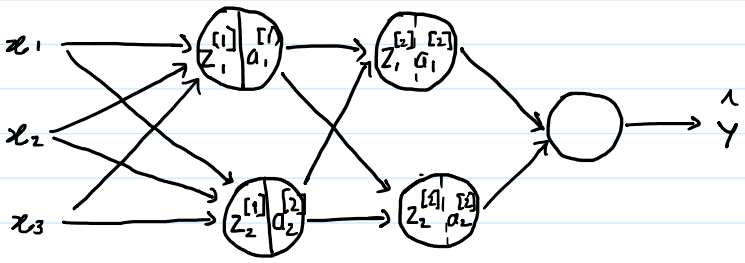
$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

γ, β: learnable parameters of the model to control mean & variance values

Use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$

Adding Batch Normalisation to a network



$$X \xrightarrow{W^{[1]}, b^{[1]}} Z^{[1]} \xrightarrow[\text{Batch Norm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{Z}^{[1]} \longrightarrow a^{[1]} = g^{[1]}(\tilde{Z}^{[1]})$$

$$\downarrow W^{[2]}, b^{[2]}$$

$$Z^{[2]}$$

$$\downarrow BN\ \beta^{[2]}, \gamma^{[2]}$$

$$a^{[2]} = g^{[2]}(\tilde{Z}^{[2]}) \longleftarrow \tilde{Z}^{[2]}$$

Parameters: $W^{[1]}, b^{[1]}, \beta^{[1]}, \gamma^{[1]}, W^{[2]}, b^{[2]}, \beta^{[2]}, \gamma^{[2]}, \ldots, W^{[L]}, b^{[L]}, \beta^{[L]}, \gamma^{[L]}$ $\quad d\beta^{[L]} \quad \beta^{[L]} = \beta^{[L]} - \alpha d\beta^{[L]}$

Working with mini batches

$$X^{\{1\}} \xrightarrow{W^{[1]}, b^{[1]}} Z^{[1]} \xrightarrow[BN]{\beta^{[1]}, \gamma^{[1]}} \tilde{Z}^{[1]} \longrightarrow a^{[2]} \longrightarrow \ldots$$

$$X^{\{2\}} \xrightarrow{W^{[1]}, b^{[1]}} Z^{[1]} \xrightarrow[BN]{\beta^{[1]}, \gamma^{[1]}} \tilde{Z}^{[1]} \longrightarrow a^{[2]}$$

Parameters: $W^{[l]}$, ~~$b^{[l]}$~~, $\beta^{[l]}$, $\gamma^{[l]}$ $\quad (n^{[l]}, 1)$

$$Z^{[l]} = W^{[l]} a^{[l-1]} + \cancel{b^{[l]}}$$

$$Z^{[l]} = W^{[l]} a^{[l-1]}$$

$$Z^{[l]}_{norm}$$

$$\tilde{Z}^{[l]} = \gamma^{[l]} Z^{[l]}_{norm} + \beta^{[l]}$$

Implement Gradient Descent

for $t = 1, \ldots, num\ Minibatches$

Compute forward prop on $X^{\{t\}}$

In each hidden layer, use BN to replace $Z^{[l]}$ with $\tilde{Z}^{[l]}$

Use backprop to compute $dW^{[l]}$, ~~$db^{[l]}$~~, $d\beta^{[l]}$, $d\gamma^{[l]}$

Update parameters $\quad W^{[l]} := W^{[l]} - \alpha dW^{[l]}$

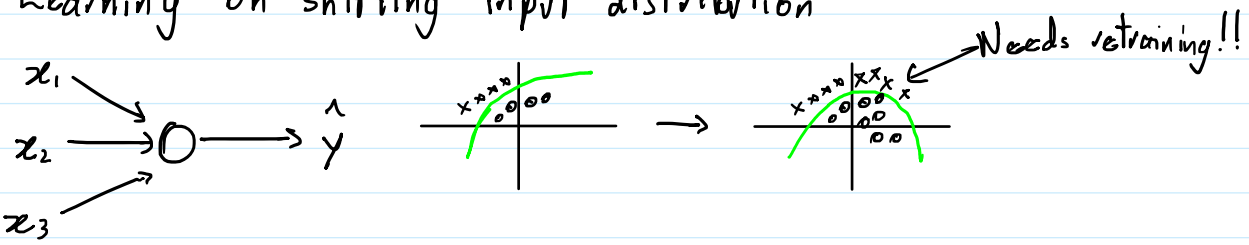
$\beta^{[l]} := \beta^{[l]} - \alpha d\beta^{[l]}$

$\gamma^{[l]} := \gamma^{[l]} - \alpha d\gamma^{[l]}$

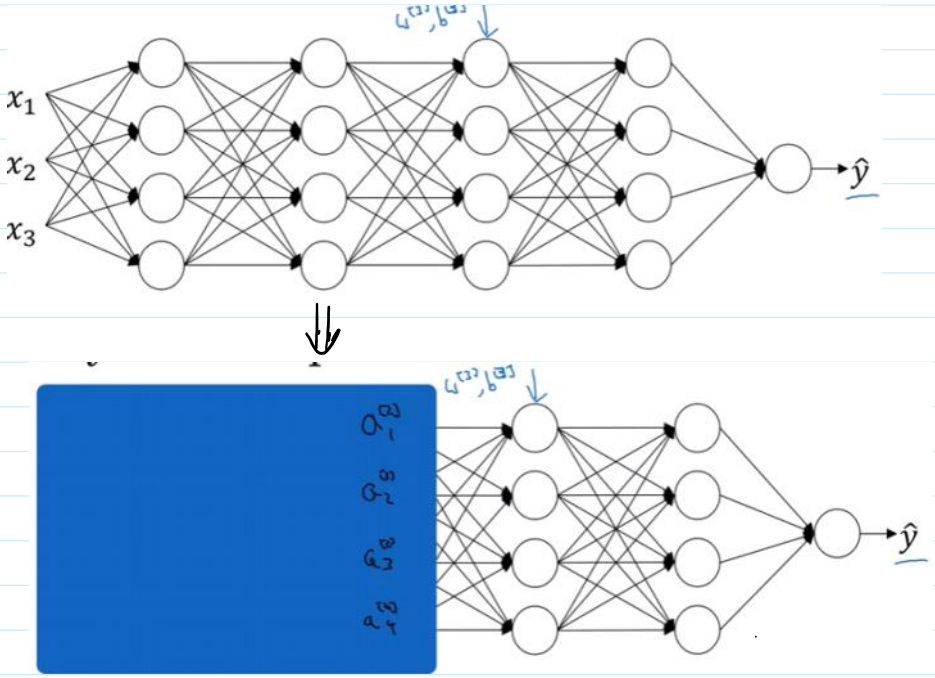
Works with momentum / RMS Prop / Adam

Why does Batch Norm work?

Learning on shifting input distribution



Cat $y=1$ Non-Cat $y=0$ Covariant Shift → $y=1$ $y=0$



Since layer 2 weights & biases are changing, $a^{[2]}$ keeps changing → covariance shift

What batch norm does is that it reduces the amount the distribution shifts

Batch Norm as a Regularisation

→ Each mini-batch is scaled by the mean/variance computed on just that mini-batch

→ This adds some noise to the values $z^{[l]}$ within that minibatch. So similar to dropout, it adds some noise to each hidden layer's activations

→ This has a slight regularisation effect

Batch Norm at Test Time

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$

→ Computed using entire mini-batch

$$z^{(i)}_{\text{norm}} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{\text{norm}} + \beta$$

μ, σ^2 : estimate using exponentially weighted average (across mini batches)

$X^{\{1\}}, X^{\{2\}}, X^{\{3\}}, \ldots$

$\mu^{\{1\}[l]} \quad \mu^{\{2\}[l]} \quad \ldots \longrightarrow \mu$

$\sigma^{2\{1\}[l]} \quad \sigma^{2\{2\}[l]} \quad \ldots \longrightarrow \sigma^2$

$$z_{norm} = \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}} \qquad \tilde{z} = \gamma z_{norm} + \beta$$

Softmax Regression

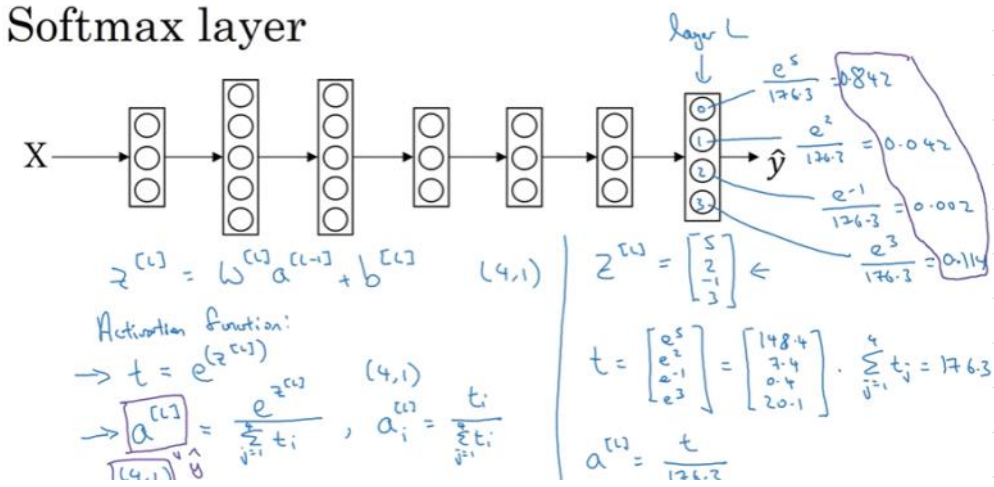
Recognising cats, dogs & baby chicks

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$

Activation function

$$\rightarrow t = e^{(z^{[l]})}$$

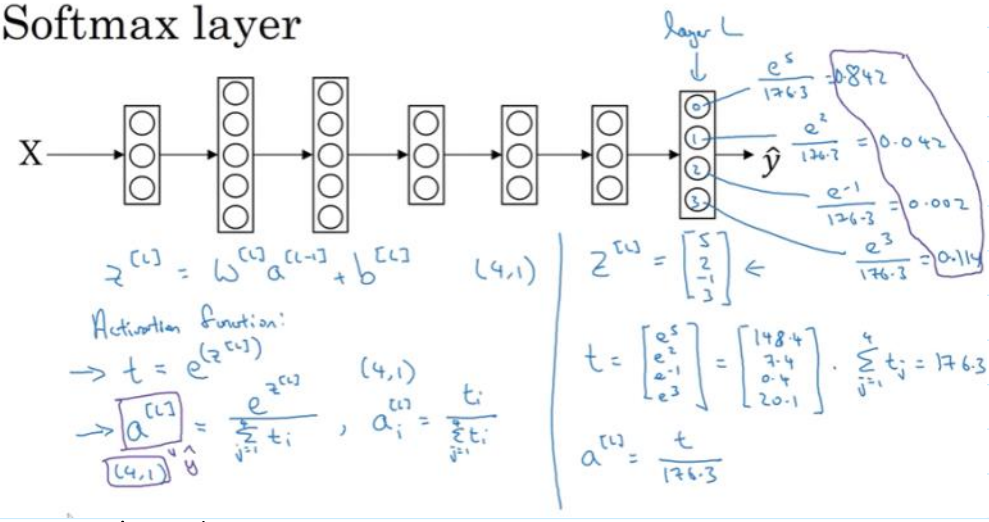
$$a_i^{[l]} = \frac{t_i}{\sum t_i}$$



$\rightarrow t = e^{(z^{[L]})}$

$a_i^{[L]} = \frac{t_i}{\sum t_i}$

Softmax layer



Training a softmax classifier

$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$ $t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$

"soft max"

$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$

"hard max"

$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

Softmax regression generalizes logistic regression to C classes

Loss function of softmax regression

$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ - cat $a^{[L]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$

$\underbrace{L(\hat{y}, y)}_{\text{small}} = -\sum_{j=1}^{4} y_j \log \hat{y}_j$

$J(w^{[L]}, b^{[L]}, \dots) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

$Y = [y^{(1)}, y^{(2)} \dots y^{(m)}]$ $\hat{y} = [\hat{y}^{(1)}, \dots, y^{(m)}]$

$= \begin{bmatrix} 0 & 0 & 1 & \dots \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix}$ $= \begin{bmatrix} 0.3 & \\ 0.2 & \\ 0.1 & \dots \\ 0.4 & \end{bmatrix}$

$$\begin{bmatrix}1 & 0 & 0 & \cdots \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix} \qquad \begin{bmatrix}0.2 & \cdots \\ 0.1 & \cdots \\ 0.4 & \end{bmatrix}$$

Backprop: $\underset{(4,1)}{dz^{[L]}} = \hat{y} - y$

$\hookrightarrow \frac{dJ}{dz^{[L]}}$

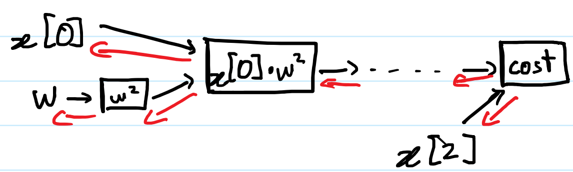
Deep learning frameworks

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

Choosing deep learning frameworks
- Ease of programming (development and deployment)
- Running speed
- Truly open (open source with good governance)

```
import numpy as np
import tensorflow as tf

w = tf.Variable(0, dtype = tf.float32)
optimizer = tf.keras.optimizers.Adam(0.1)
def training_step():
        with tf.GradientTape() as tape:
                cost = w**2 - 10 * w + 25
        trainable_variables = [w]
        grads = tape.gradient(cost, trainable_variables)
        optimizer.apply_gradients(zip(grads, trainable_variables))

print(w)

train_step()
print(w)

for i in range(1000):
        train_step()
print(w)

w = tf.Variable(0, dtype = tf.float32)
x = np.array([1.0, -10.0, 25.0], dtype = np.float32)
optimizer = tf.keras.optimizers.Adam(0.1)

def training(x, w, optimizer):
        def cost_fn():
                cost = x[0]*w**2 + x[1] * w + x[2]
                return cost
        for i in range(1000):
                optimizer.minimize(cost_fn, [w])
        return w

print(training(x, w, optimizer))
```

Computation graph constructed



Transfer Learning with Tensorflow Hub

20 October 2021 08:46

https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub

To recognise class in labels that have been trained on

```
import numpy as np
import time

import PIL.Image as Image
import matplotlib.pylab as plt

import tensorflow as tf
import tensorflow_hub as hub

import datetime

%load_ext tensorboard

mobilenet_v2 ="https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/4"
inception_v3 = "https://tfhub.dev/google/imagenet/inception_v3/classification/5"

classifier_model = mobilenet_v2

IMAGE_SHAPE = (224, 224)

classifier = tf.keras.Sequential([
    hub.KerasLayer(classifier_model, input_shape=IMAGE_SHAPE+(3,))
])

grace_hopper =
tf.keras.utils.get_file('image.jpg','https://storage.googleapis.com/download.tensorflow.org/example_images/grace_hopper.jpg')
grace_hopper = Image.open(grace_hopper).resize(IMAGE_SHAPE)
grace_hopper

grace_hopper = np.array(grace_hopper)/255.0
grace_hopper.shape

# Add a batch dimension with np.newaxis and pass the image to the model
result = classifier.predict(grace_hopper[np.newaxis, ...])
result.shape

predicted_class = tf.math.argmax(result[0], axis = -1)
predicted_class

labels_path =
tf.keras.utils.get_file('ImageNetLabels.txt','https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt')
imagenet_labels = np.array(open(labels_path).read().splitlines())

plt.imshow(grace_hopper)
plt.axis('off')
predicted_class_name = imagenet_labels[predicted_class]
_ = plt.title("Prediction: " + predicted_class_name.title())
```

To predict labels of a class different from trained labels

But what if you want to create a custom classifier using your own dataset that has classes that aren't included in the original ImageNet dataset (that the pre-trained model was trained on)?

To do that, you can:

1. Select a pre-trained model from TensorFlow Hub; and
2. Retrain the top (last) layer to recognize the classes from your custom dataset.

```
data_root = tf.keras.utils.get_file( 'flower_photos',  'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',
untar=True)
batch_size = 32
img_height = 224
img_width = 224

train_ds = tf.keras.utils.image_dataset_from_directory(str(data_root),
                                                       validation_split = 0.2,
```

```
                                                    subset = 'training',
                                                    seed = 123,
                                                    image_size = (img_height, img_width),
                                                    batch_size = batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(str(data_root),
                                                  validation_split = 0.2,
                                                  subset = 'validation',
                                                  seed = 123,
                                                  image_size = (img_height, img_width),
                                                  batch_size = batch_size)

class_names = np.array(train_ds.class_names)
print(class_names)

normalization_layer = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: normalization_layer(x), y)
val_ds = val_ds.map(lambda  x, y: normalization_layer(x), y)

AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break

#Now download the headless model

mobilenet_v2 = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
inception_v3 = "https://tfhub.dev/google/tf2-preview/inception_v3/feature_vector/4"

feature_extractor_model = mobilenet_v2

feature_extractor_layer = hub.KerasLayer(feature_extractor_model, input_shape = (224, 224, 3), trainable = False)

num_classes = len(class_names)

model = tf.keras.Sequential([feature_extractor_layer, tf.keras.layers.Dense(num_classes)])

model.summary()

predictions = model(image_batch)

predictions.shape

model.compile(optimizer = tf.keras.optimizers.Adam(), loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True ), metrics = ['accuracy'])

# Enable histogram computation for every epoch
log_dir = "logs/fit/" + datetime.datetime.now().strftime('%Y%m%d - %H%M%S')
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir = log_dir, histogram_freq = 1)

NUM_EPOCHS = 10

history = model.fit(train_ds, validation_data = val_ds,  epochs = NUM_EPOCHS, callbacks = tensorboard_callback)

# Start the Tensorboard to view how the metrics change with each epoch and to track other scalar values
%tensorboard --logdir logs/fit

#Check the predictions
predicted_batch = model.predict(image_batch)
predicted_id = tf.math.argmax(predicted_batch, axis=-1)
predicted_label_batch = class_names[predicted_id]
print(predicted_label_batch)

#Plot the model predictions
plt.figure(figsize = (10,9))
for n in range(30):
        plt.subplot(6, 5, n+1)
        plt.imshow(image_batch[n])
        plt.title(predicted_label_batch[n].title())
        plt.axis('off')

_ = plt.suptitle('Model Predictions')
```

Transfer Learning and fine-tuning

20 October 2021 09:28

https://www.tensorflow.org/tutorials/images/transfer_learning

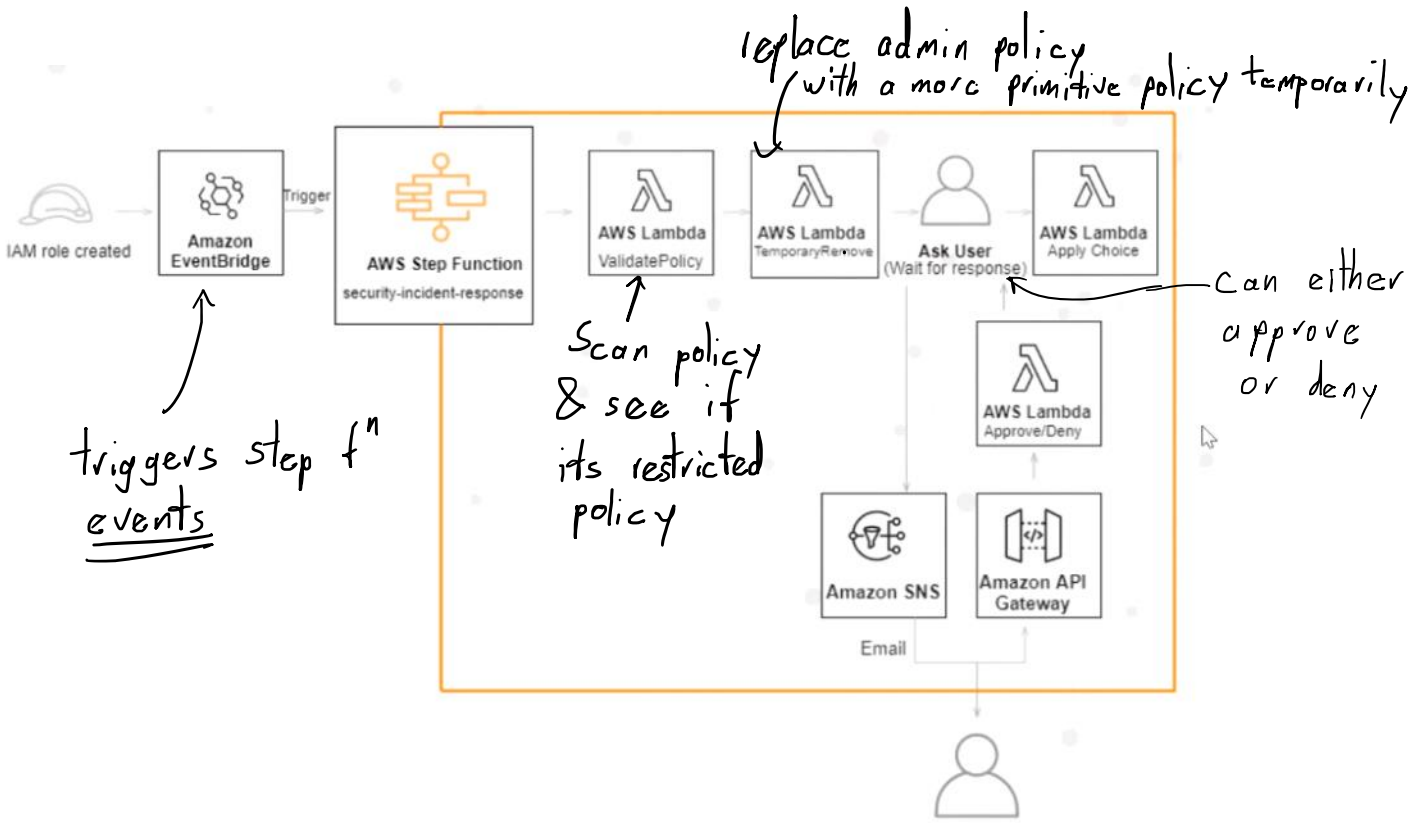
```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf

from tensorflow.keras.preprocessing import image_dataset_from_directory

_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

BATCH_SIZE = 32
IMG_SIZE = (160, 160)

train_dataset = image_dataset_from_directory(train_dir,
                                             shuffle=True,
                                             batch_size=BATCH_SIZE,
                                             image_size=IMG_SIZE)

validation_dataset = image_dataset_from_directory(validation_dir,
                                                  shuffle=True,
                                                  batch_size=BATCH_SIZE,
                                                  image_size=IMG_SIZE)

class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")

val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)

AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)

data_augmentation = tf.keras.Sequential([tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
tf.keras.layers.experimental.preprocessing.RandomRotation(0.2)])

for image, _ in train_dataset.take(1):
        plt.figure(figsize = (10,10))
```

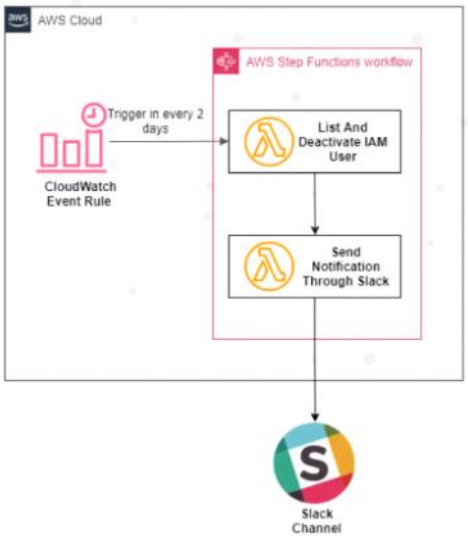
```
        first_image = image[0]
        for i in range(9):
                ax = plt.subplot(3,3, i+1)
                augmented_image = data_augmentation(tf.expand_dims(first_image, 0))
                plt.imshow(augmented_image[0]/255)
                plt.axis('off')

preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input

rescale = tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset = -1)

IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape = IMG_SHAPE,
                                               include_top = False,
                                               weights = 'imagenet')

image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)

base_model.trainable = False

base_model.summary()

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)

prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)

inputs = tf.keras.Input(shape = (160,160,3))
x = data_augmentation(inputs)
x = base_model(x, trainable = False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

base_learning_rate = 0.001
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = base_learning_rate), loss =
tf.keras.losses.BinaryCrossentropy(from_logits = True), metrics = ['accuracy'])

model.summary()

len(model.trainable_variables)

initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)

print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))

history = model.fit(train_dataset, epochs = initial_epochs, validation_data = validation_dataset)
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()

# Now that we have trained the bottom layers, let us unfreeze a few top layers and freeze the bottom layers

base_model.trainable = True

# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(lr=base_learning_rate/10),
              metrics=['accuracy'])

model.summary()

len(model.trainable_variables)

fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)

acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
```

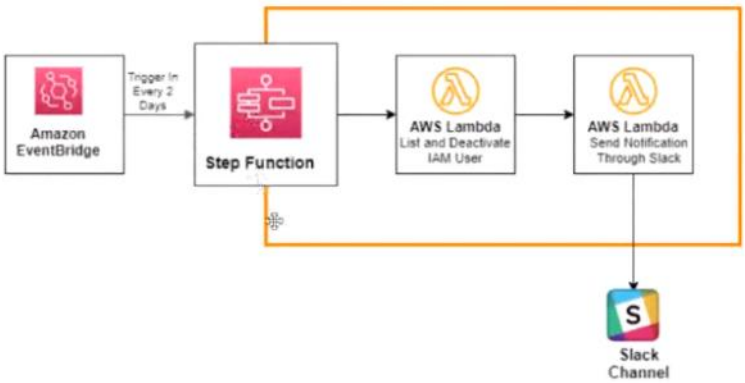
```
val_loss += history_fine.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()

loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)

# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tf.nn.sigmoid(predictions)
predictions = tf.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(9):
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

14 September 2021 16:08

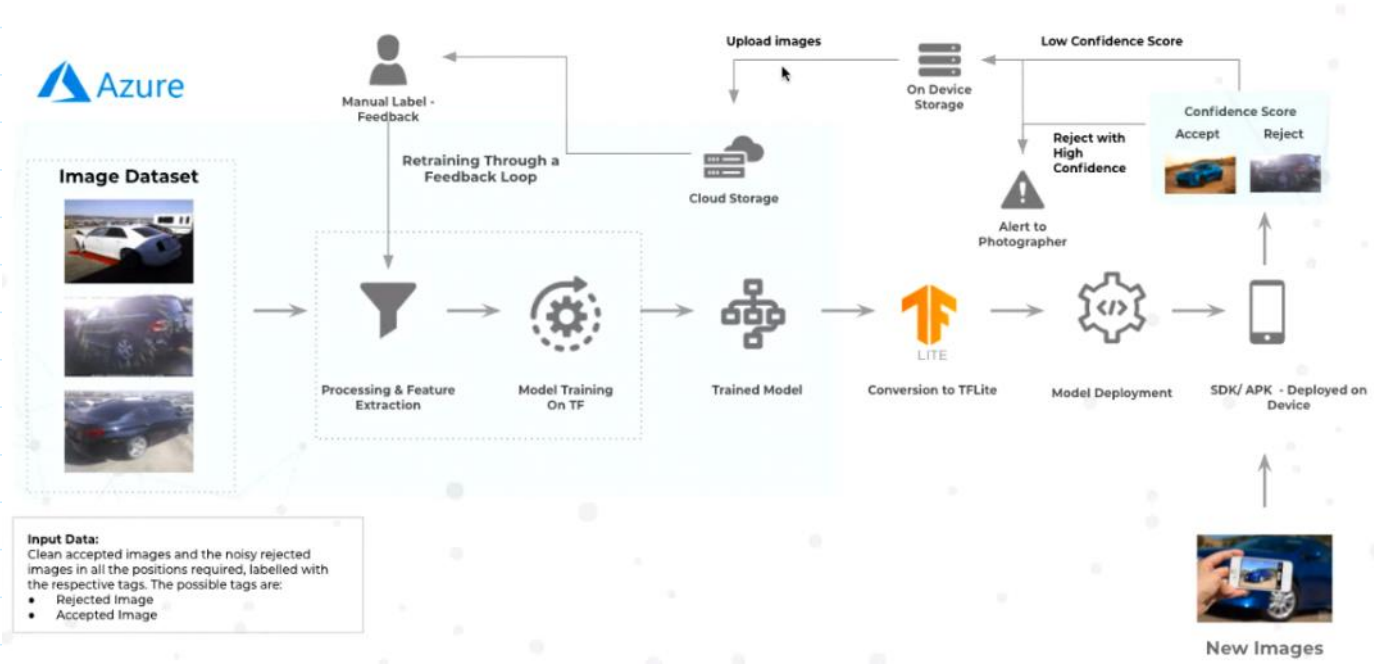




Additional usecase



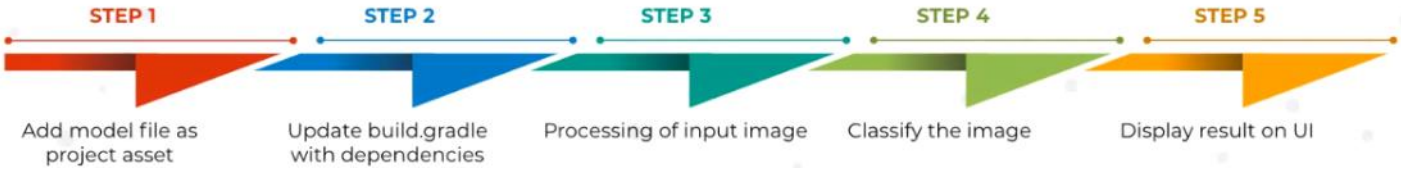
23 September 2021 16:04



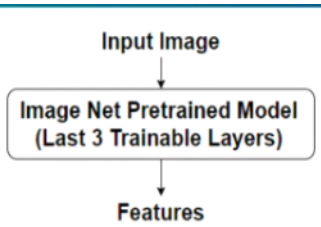
Creation of Android SDK as AAR file and deploy on edge device

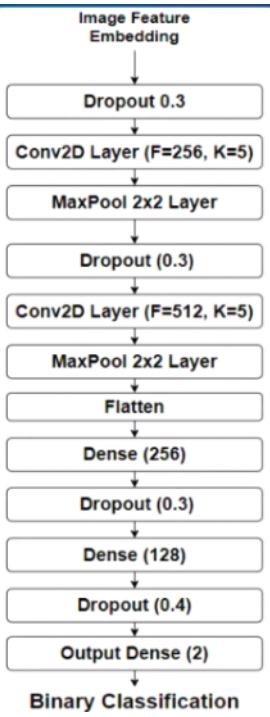
- Adding dependencies with the project (example tflite interpreter , kotlin coroutines etc)
- Android library compiles into an Android Archive (AAR) file that you can use as a dependency for an Android app module

Model Deployment on Android Device



- Accuracy : 87.3
- Precision : 82.3
- Recall : 76.7
- F1 score : 79.4
- Latency : Less than 1 second
- Pretrained Model - MobileNet V1

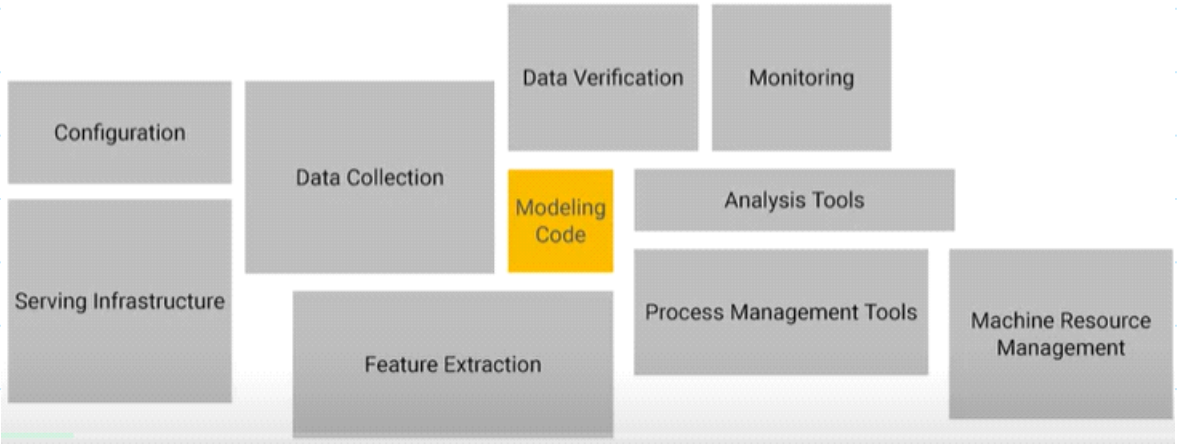


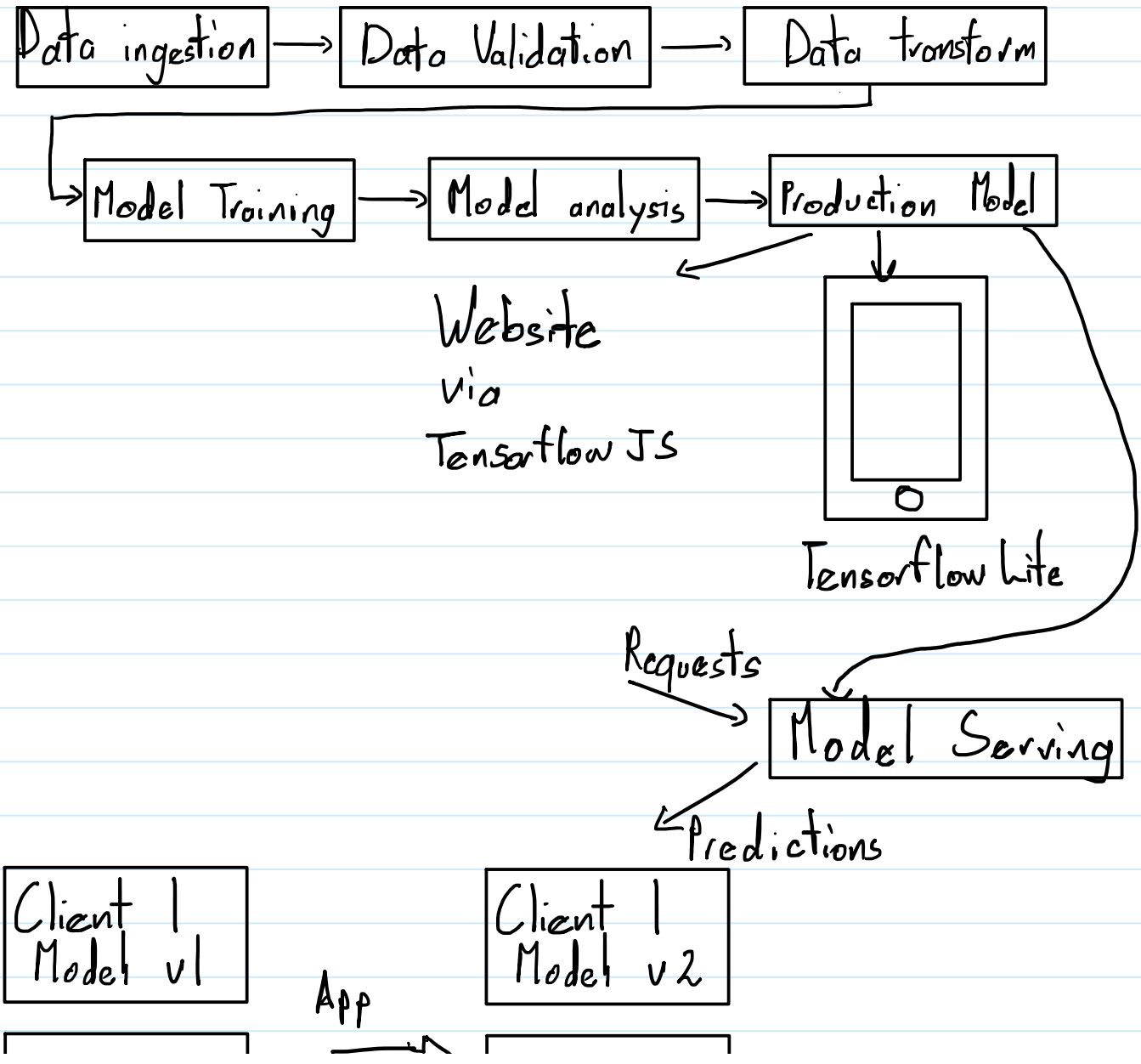


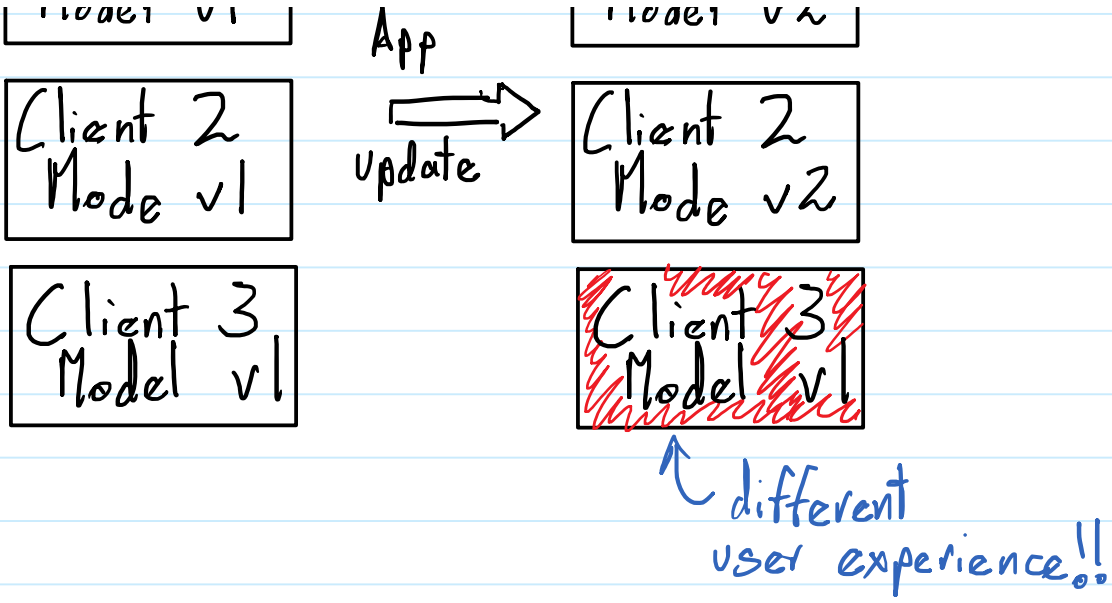
TF Serving

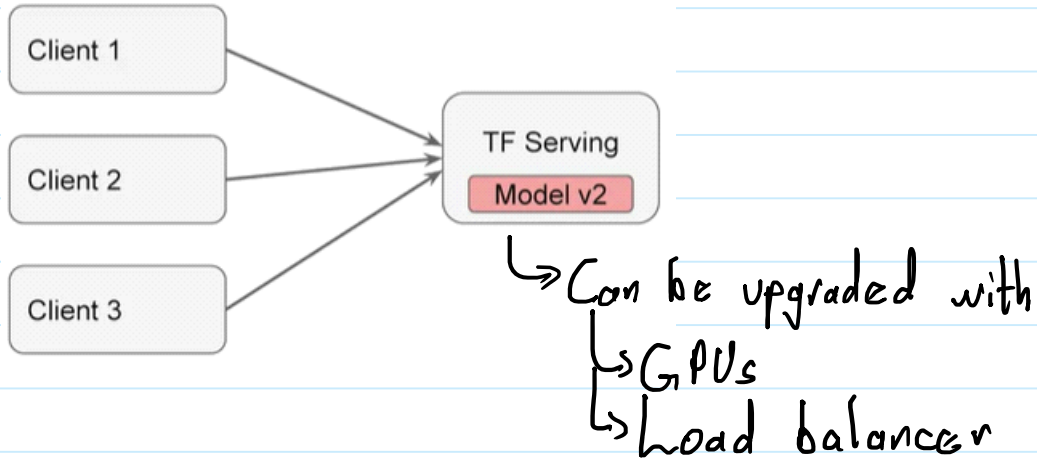
28 October 2021 11:45

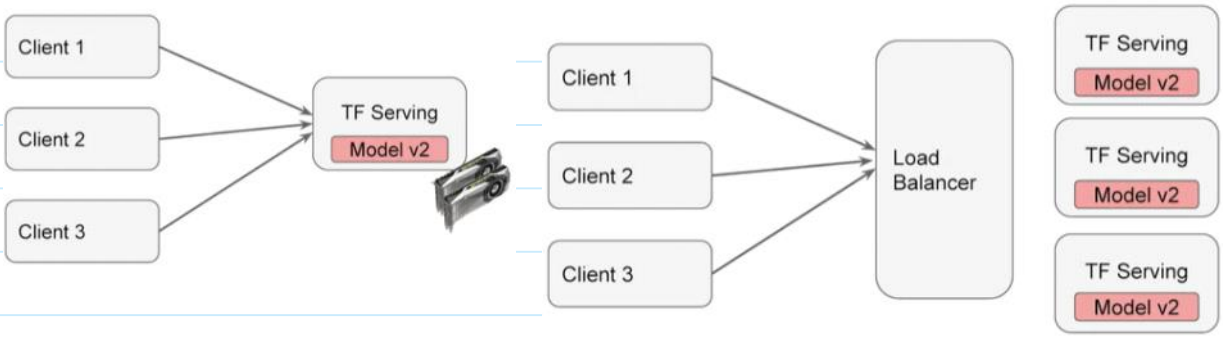
Building Models is just a small part of ML











Tensorflow Serving

```
import tensorflow as tf
import numpy as np
from tensorflow import keras
model = tf.keras.Sequential([keras.layers.Dense(units = 1, input_shape[1])])
model.compile(optimizer = 'sgd', loss = 'mean_squared_error')
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype = float)
xs = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype = float)

model.fit(xs, ys, epochs = 500, verbose = 2)
```

```
print(model.predict([10.0]))

import tempfile
MODEL_DIR = tempfile.gettempdir()
version = 1
export_path = os.path.join(MODEL_DIR, str(version))
if os.path.isdir(export_path):
        print('Already saved a model, cleaning up')
        !rm -r {export_path}

tf.saved_model.simple_save(keras.backend.get_session(), export_path, inputs = {'input_image':
model.input}, outputs = {t.name:t for t in model.outputs})

print('\nSaved model:')
!ls -l {export_path}

!saved_model_cli show --dir {export_path} --all

os.environment['MODEL_DIR'] = MODEL_DIR

%%bash --bg
nohup tensorflow_model_server \
        --rest_api_port = 8501 \
        --model_name = helloworld \
        --model_base_path="${MODEL_DIR}" >server.log 2>&1

!tail server.log

import json
xs = np.array([[9.0],[10.0]])
data = json.dumps({"signature_name": "serving_default", "instances": xs_tolist()})
print(data)

import requests
headers = {"content-type": "application/json"}
json_response = requests.post('http://localhost:8501/v1/models/helloword:predict',
                                        data = data,
                                         headers = headers)
print(json_response.text)
predictions = json.loads(json_response.text)['predictions']
```

But what about more complex models ??

```
import sys

assert sys.version_info.major is 3

print('Installing dependencies for Colab environment')
!pip install -Uq grpcio==1.26.0

import tensorflow as tf
from tensorflow import keras

import numpy as np
```

```
import matplotlib.pyplot as plt
import os
import subprocess

print('Tensorflow version: {}'.format(tf.__version__))
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

train_images = train_images/255.0
test_images = test_images/255.0

train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
test_images = test_images.reshape(train_images.shape[0], 28, 28, 1)

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle
boot']

print('\ntrain_images.shape: {}, of {}'.format(train_images.shape, train_images.dtype))
print('test_images.shape: {}, of {}'.format(test_images.shape, test_images.dtype))

model = keras.Sequential([keras.layers.Conv2D(input_shape = (28, 28,1), filters = 8, kernel_size = 3,
strides = 2, activation = 'relu', name = 'Conv1'), keras.layers.Flatten(), keras.layers.Dense(10, name =
"Dense")])
model.summary()
testing = False
epochs = 5

model.compile(optimizer = 'adam', loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits =
True), metrics = [keras.metrics.SparseCategoricalAccuracy()])

model.fit(train_images, train_labels, epochs = epochs)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('\nTest accuracy: {}'.format(test_acc))

import tempfile
MODEL_DIR = tempfile.gettempdir()
version = 1
export_path = os.path.join(MODEL_DIR, str(version))
print('export_path = {}\n'.format(export_path))
tf.keras.models.save_model(model, export_path, overwrite = True, include_optimizer = True,
save_format = None, signatures = None, options = None)
print('\nSaved model:')
!ls -l {export_path}

saved_model_cli show --dir {export_path} --all

import sys

if 'google.colab' not in sys.modules:
        SUDO_IF_NEEDED = 'sudo'
else:
        SUDO_IF_NEEDED = ''

# This is the same as you would do from your command line, but without the [arch=amd64], and
no sudo
# You would instead do:
```

```
# echo "deb [arch=amd64] http://storage.googleapis.com/tensorflow-serving-apt stable
tensorflow-model-server tensorflow-model-server-universal" | sudo tee
/etc/apt/sources.list.d/tensorflow-serving.list && \
# curl https://storage.googleapis.com/tensorflow-serving-apt/tensorflow-serving.release.pub.gpg
| sudo apt-key add -

!echo "deb http://storage.googleapis.com/tensorflow-serving-apt stable tensorflow-model-server
tensorflow-model-server-universal" | {SUDO_IF_NEEDED} tee /etc/apt/sources.list.d/tensorflow-
serving.list && \
curl https://storage.googleapis.com/tensorflow-serving-apt/tensorflow-serving.release.pub.gpg |
{SUDO_IF_NEEDED} apt-key add -
!{SUDO_IF_NEEDED} apt update

{SUDO_IF_NEEDED} apt-get install tensorflow-model-server

os.environ['MODEL_DIR'] = MODEL_DIR

nohup tensorflow_model_server \
        --rest_api_port = 8501 \
        --model_name = fashion_model \
        --model_base_path = '${MODEL_DIR}' >server.log 2>&1

tail server.log

def show(idx, title):
        plt.figure()
        plt.imshow(test_images[idx].reshape(28,28,1))
        plt.axis('off')
        plt.title('\n\n{}'.format(title), fontdict = {'size': 16})

import random
rando = random.randint(0,len(test_images) - 1)
show(rando, 'An Example Image: {}'.format(classes[test_labels[rando]]))

import json
data = json.dumps({'signature_name': 'serving_default', 'instances': test_images[0:3].tolist()})
print('Data: {} ... {}'.format(data[:50], data[len(data)-52:]))

!pip install -q requests
import requests
headers = {'content-type':'applications/json'}
json_response = requests.post('https://localhost:8501/v1/models/fashion_model:predict', data = data,
headers = headers)

predictions = json.loads(json_response.text)['predictions']

show(0, 'The model thought this was a {} (class {}), and it was actually a {} (class {})'.format(
  class_names[np.argmax(predictions[0])], np.argmax(predictions[0]), class_names[test_labels[0]],
test_labels[0]))
```

Tensorflow Extended

27 October 2021 09:57

In addition to training the model:-

TFX
→ To build production pipelines
→ Powers the most important products & projects

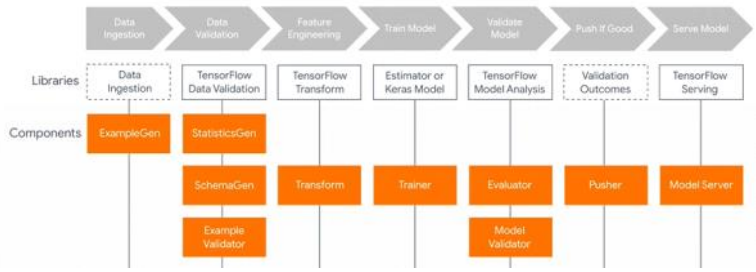
Machine Learning Development

- Labelled data
- Feature space coverage
- Minimal dimensionality
- Maximum predictive data
- Fairness
- Rare Conditions
- Data Lifecycle Management

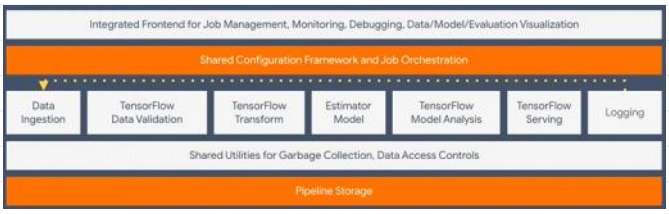
Modern Software Development

- Scalability
- Extensibility
- Configuration
- Consistency & reproducibility
- Modularity
- Best practices
- Testability
- Monitoring
- Safety & Security

TFX Production Components



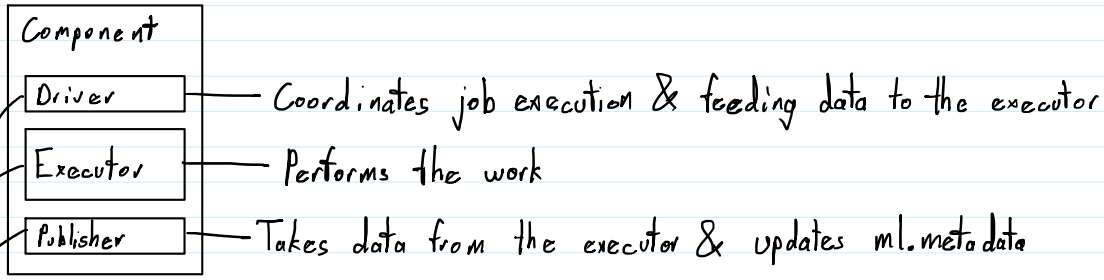
Horizontal Layers Coordinate Components



How do pipelines work?

What is a component?

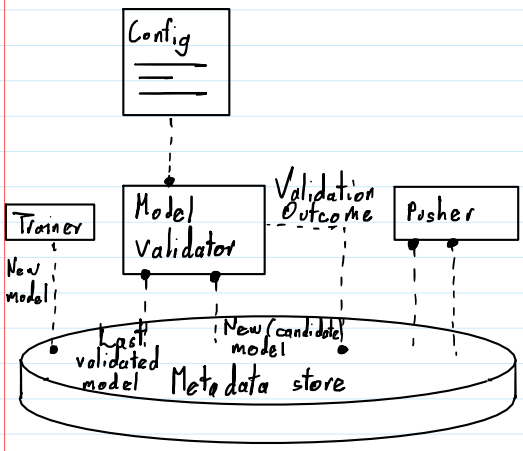
- Pipelines created as a sequence of components
- Each component performs a different task
- Components organised into Directed Acyclic Graphs



Boilerplate Code → Could be changed but you don't need to in most cases

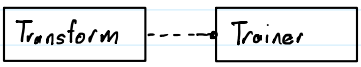
Where you insert your code & do your customisations

What makes a component?

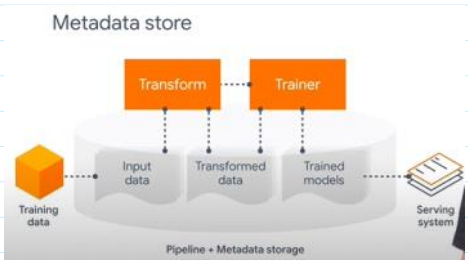


Orchestration Styles :-

Task Aware Pipelines



Task & Data Aware Pipelines



Stores all artifacts of every component over many executions

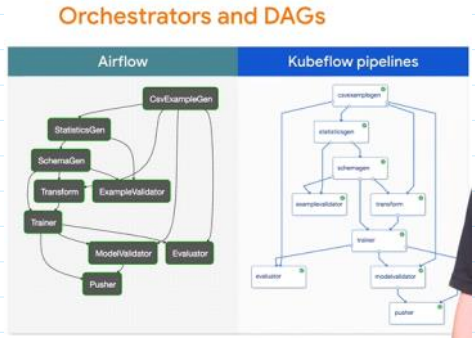
TFX Orchestration

TFX Orchestration

→ To put an ML pipeline together, to find the sequence of components that make up the pipeline and manage their execution we need an orchestrator

→ Orchestrator can be used to trigger tasks & monitor our components

Bring your own Orchestrator!



Why do I need metadata?

→ TFX implements metadata storage using ML Metadata (open source framework)

→ Stored in a Relational Database - any SQL compatible database will do

Whats in metadata store?

→ Type definitions of artifacts & their properties

Trained models	Data that we train models on	Evaluation Results

Things stored in metadata → artifacts

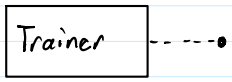
Artifacts have properties

Data itself is stored outside database

↳ Properties & location of the data object is kept in the meta data store

Execution Records (runs) of components

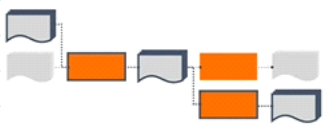
Execution Records (runs) of components



ML pipeline is run frequently over a long lifetime
- New data comes in
- Conditions change

} keeping history becomes important

Data provenance across all executions



Allows us to track forward & backward through the pipeline to understand the origins & results of running components

Metadata-powered functionality

→ Find out which data, a model was trained on
→ Impact some new feature engineering had on our evaluation metrics
In some cases may even be a regulatory requirement

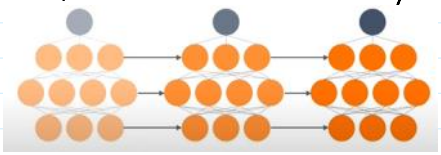
Its not just for today's model or today's results, we're interested in understanding how our data or results change with time

Compare with model runs in the past

Production solutions aren't a one time thing

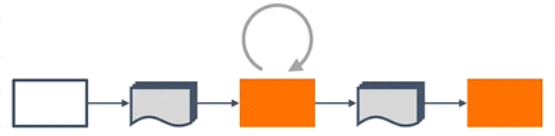
We can make our pipelines much more efficient by only rerunning component when necessary

Carry over state from previous runs



Only possible if we have saved our model in metadata

Re-use previously computed outputs



Rerun only when input or code has changed

Pull previous result from cache

If a new run only changes the parameters of the trainer, the pipeline can reuse data preprocessing artifacts such as vocabularies & this can save a lot of time given that large data volumes make preprocessing expensive

trainer, the pipeline can reuse data preprocessing artifacts such as vocabularies & this can save a lot of time given that large data volumes make preprocessing expensive

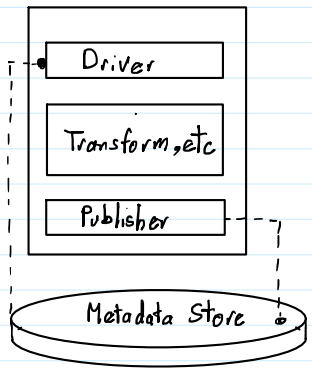
TFX & ml. metadata enables this reusability out of the box

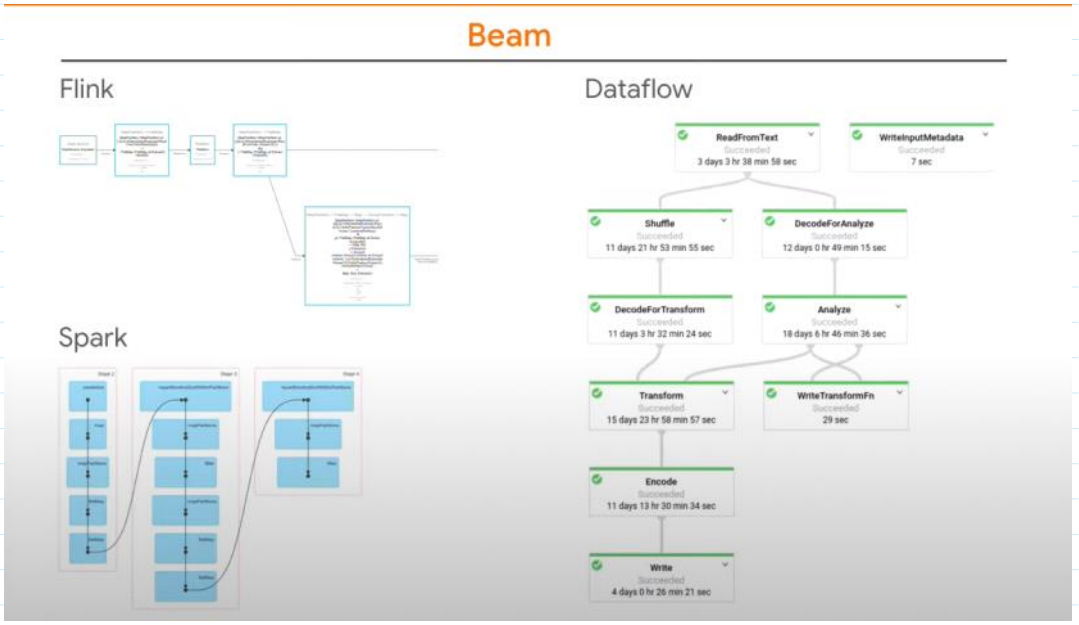
Distributed processing & components

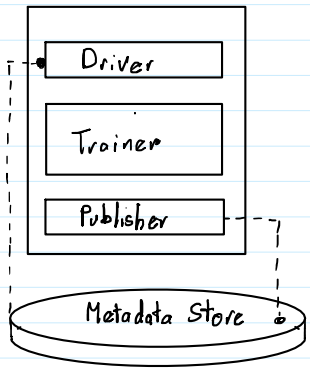
Apache Beam

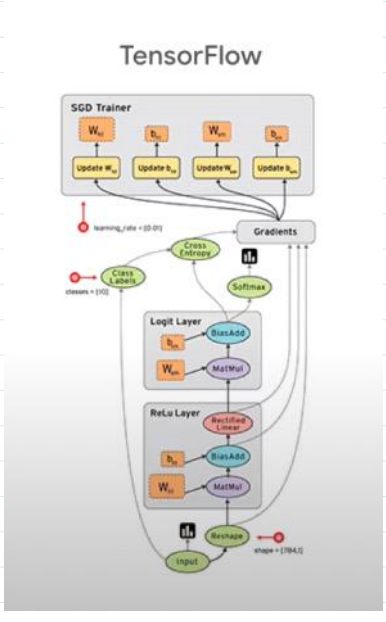
- To handle distributed processing of large amounts of data especially compute intensive data like ML workloads
- A unified batch & stream distributed processing API
- A set of SDK frontends: Java, Python, Go, Scala, SQL
- A set of Runners which can execute Beam jobs into various backends: Local, Apache, Flink, Apache Spark Apache Gearpump, Apache Samza, Apache Hadoop, Google Cloud Dataflow, ...

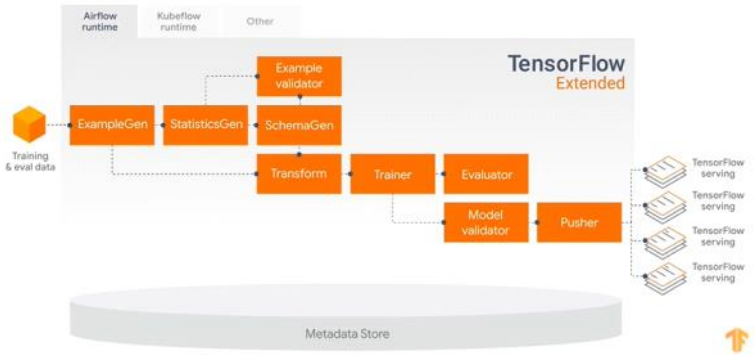
Executors do the work







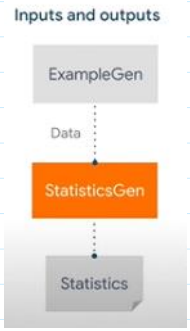




Component: ExampleGen


```
examples = csv_input(os.path.join(data_root, 'simple'))
example_gen = CsvExampleGen(input_base = examples)
```

Component: Statistics Gen




```
statistics_gen = StatisticsGen(input_data = example_gen.outputs.examples)
```

↳ Explore & Understand data & find issues
↳ Uses Tensorflow data validation library

Component: Schema Gen


```
infer_schema = SchemaGen(stats = statistics_gen.outputs.output)
```

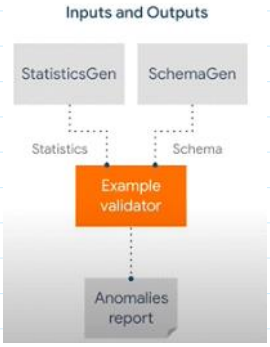
↳ Uses Tensorflow Data Validation Library
↳ Looks at the statistics generated by Statistics Gen & tries to infer the types for each of our features
↳ Can be adjusted as needed (like adding new categories

Feature name	Type	Presence	Valency	Domain
'fare'	FLOAT	required	single	-
'trip_start_hour'	INT	required	single	-
'pickup_census_tract'	BYTES	optional		-
'dropoff_census_tract'	FLOAT	optional	single	-
'company'	STRING	optional	single	'company'

Component: Example Validator


```
validate_stats = ExampleValidator(stats = statistics_gen.outputs.output,
                                  schema = infer_schema.outputs.output)
```

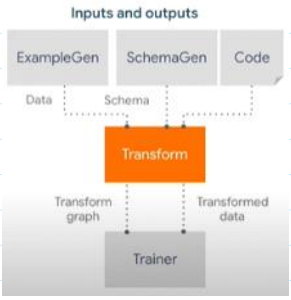
Component: ExampleValidator



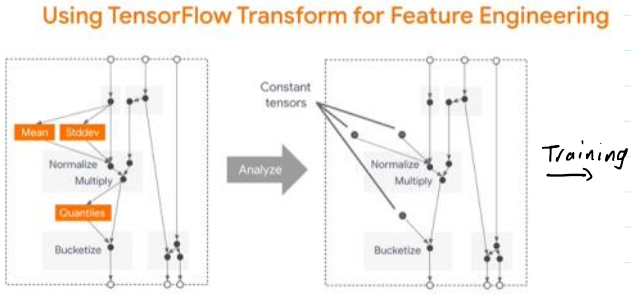

```
validate_stats = ExampleValidator(stats = statistics_gen.outputs.output,
                                  schema = infer_schema.outputs.output)
```

↳ Takes statistics from Statistics Gen & schema from Schema Gen — may be the outputs of schema gen or the results of user curation & looks for problems, anomalies, missing values or values that don't match our schema

Component: Transform




```
transform = Transform(input_data = example_gen.outputs.examples,
                      schema = infer_schema.outputs.output,
                      module_file=taxi_module_file)

for key in _DENSE_FLOAT_FEATURE_KEYS:
    outputs[_transformed_name[key]] = transform.scale_to_z_score(_fill_in_missing(inputs[key]))

outputs[_transformed_name[_LABEL_KEY] = tf.where(
    tf.is_nan(taxi_fare),
    tf.cast(tf.zeros_like(taxi_fare), tf.int64),
    tf.cast(tf.greater(tips, tf.multiply(taxi_fare, tf.constant(0.2), tf.int64))))]
```



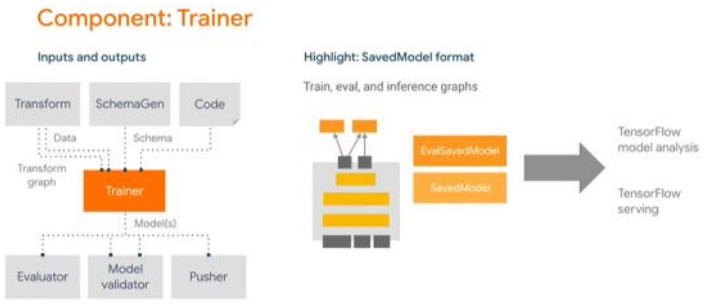
Transform will do a complete pass over data — a full epoch & create 2 different kinds of results

For calculating the median or standard deviation :-
↳ Transform outputs a constant

For normalizing a value :-
↳ Transform outputs tf.ops

Transform outputs tf graphs with those constants & ops

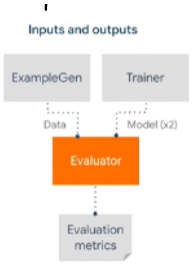
Eliminates training/serving skew




```
trainer = Trainer(module_file = taxi_module_file,
                  transformed_examples = transform.outputs.transformed_examples,
                  schema =infer_schema.outputs.output,
                  transform_output = transform.outputs.transform_output,
                  train_steps = 10000,
                  eval_steps = 5000,
                  warm_starting = True)
```

Component : Evaluator

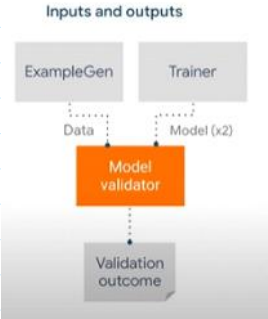

```
model_analyzer = Evaluator(examples = examples_gen.outputs.output,
                           eval_spec = taxi_eval_spec,
                           model_exports = trainer.outputs.output)
```




```
model_analyzer = Evaluator(examples = examples_gen.outputs.output,
                           eval_spec = taxi_eval_spec,
                           model_exports = trainer.outputs.output)
```

→ It looks at not only the top level results across the whole training set but also the individual slices - because the experience of each user of our model depends on the individual datapoint.

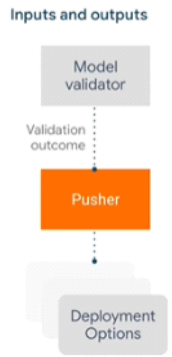
Component: Model Validator




```
model_validator = ModelValidator(examples = examples_gen.outputs.output,
                                 model = trainer.outputs.output,
                                 eval_spec = taxi_mv_spec)
```

Configuration options
1. Validate using current eval data
2. Next-day eval: Validate using unseen data

Component: Pusher




```
pusher = Pusher(model_export = trainer.outputs.output,
                model_blessing = model_validator.outputs.blessing,
                serving_model_dir = serving_model_dir)
```

Block push on validation outcome
Push destinations supported today
1. Filesystem (Tensorflow Lite, TensorflowJS)
2. Tensorflow Serving

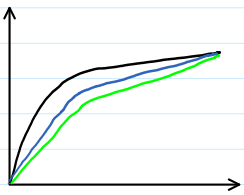
Model understanding & business reality

Online retailer selling shoes

Your model predicts click through rates helping you decide how much inventory to order

All of a sudden!

AUC & prediction accuracy have dropped on men's dress shoes



Why "understand" the model?

→ Mispredictions do not have uniform cost to your business

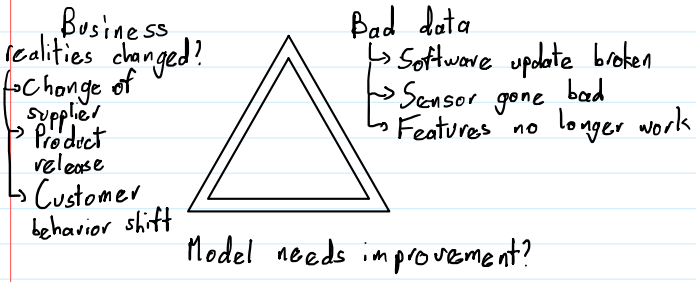
→ The data that you have is rarely you wish you had

→ Model objective is nearly always a proxy for your business objective

→ Real world doesn't stand still

ML Insights Triangle

Some assumption was violated..... but which one?



First things first
Start with your data

Check your data with the ExampleValidator component & the tools in Tensorflow data validation
- No outliers
- No missing features
- Minimal distribution shift

Feature attributions
Query for other examples by matching on those important features
→ Maybe the model generalized too few examples with this particular feature combo?
→ Add features to help create distinctions you'd like your model to make.
→ Collect more examples with that feature combo if possible!

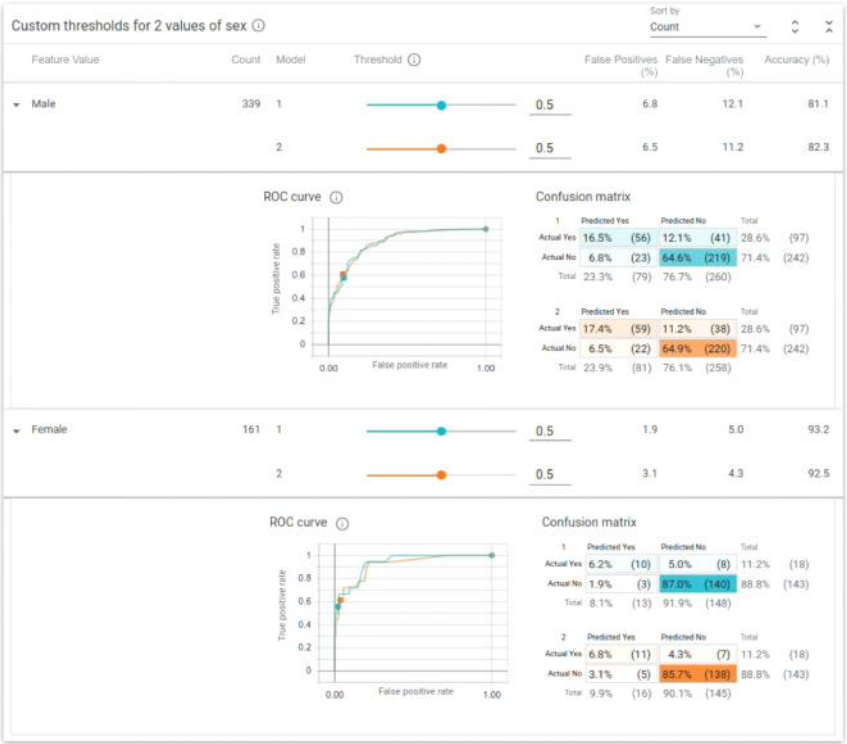
Analyse & compare

Check your model performance with the Evaluator component and the tools in Tensorflow Model Analysis
→ How does the model perform on different slices of data?
→ How does the current model performance compare to the previous versions

Explore your model and data

What-if tool

- Understand the input your model is receiving
- Ask and answer “what-if” questions about your model’s output
- Compare model performance across different slices of your data
- Compare performance across multiple models



Simple TFX Pipeline Tutorial using Penguin Dataset

28 October 2021 10:38

https://www.tensorflow.org/tfx/tutorials/tfx/penguin_simple

```
!pip install -U tfx
import tensorflow as tf
from tfx import v1 as tfx

import os

PIPELINE_NAME = 'penguin-simple'
# Output directory to store artifacts generated from the pipeline
PIPELINE_ROOT = os.path.join('pipelines', PIPELINE_NAME)

# Path to a SQLite DB file to use as an MLMD storage
METADATA_PATH = os.path.join('metadata', PIPELINE_NAME, 'metadata.db')

# Output directory where created models from the pipeline will be exported
SERVING_MODEL_DIR = os.path.join('serving_model', PIPELINE_NAME)

from abs1 import logging
logging.set_verbosity(logging.INFO) # Set default logging level

import urllib.request
import tempfile

DATA_ROOT = tempfile.mkdtemp(prefix = 'tfx-data') #Create a temporary directory
_data_url = 'https://raw.githubusercontent.com/tensorflow/tfx/master/tfx/examples/penguin/data/labelled/penguins_processed.csv'
_data_filepath = os.path.join(DATA_ROOT, 'data.csv')
urllib.request.urlretrieve(_data_url, _data_filepath)

_trainer_module_file = 'penguin_trainer.py'
%%writefile {_trainer_module_file}
from typing import List
from abs1 import logging
import tensorflow as tf
from tensorflow import keras
from tensorflow_transform.tf_metadat import schema_utils

from tfx import v1 as tfx
from tfx_bs1.public import tfxio
from tensorflow_metadat.proto.v0 import schema_pb2

_FEATURE_KEYS = ['culmen_length_mm', 'culmen_depth_mm', 'flipper_length_mm', 'body_mass_g']
_LABEL_KEY = 'species'

_TRAIN_BATCH_SIZE = 20
_TEST_BATCH_SIZE = 10

_FEATURE_SPEC = {
      **{
            feature:tf.io.FixedLenFeature(shape = [1], dtype = tf.float32)
            for feature in _FEATURE_KEYS
        }
      _LABEL_KEY: tf.io.FixedLenFeature(shape = [1], dtype = tf.int64)
}

def _input_fn(file_pattern: List[str],
          data_accessor: tfx.components.DataAccessor,
          schema: schema_pb2.Schema,
          batch_size: int = 200) -> tf.data.Dataset:
```

```
  """Generates features and label for training.

  Args:
    file_pattern: List of paths or patterns of input tfrecord files.
    data_accessor: DataAccessor for converting input to RecordBatch.
    schema: schema of the input data.
    batch_size: representing the number of consecutive elements of returned
      dataset to combine in a single batch

  Returns:
    A dataset that contains (features, indices) tuple where features is a
      dictionary of Tensors, and indices is a single Tensor of label indices.
  """
  return data_accessor.tf_dataset_factory(
      file_pattern,
      tfxio.TensorFlowDatasetOptions(
          batch_size=batch_size, label_key=_LABEL_KEY),
      schema=schema).repeat()


def _build_keras_model() -> tf.keras.Model:
  """Creates a DNN Keras model for classifying penguin data.

  Returns:
    A Keras Model.
  """
  inputs = [keras.layers.Input(shape=(1,), name=f) for f in _FEATURE_KEYS]
  d = keras.layers.concatenate(inputs)
  for _ in range(2):
    d = keras.layers.Dense(8, activation='relu')(d)
  outputs = keras.layers.Dense(3)(d)

  model = keras.Model(inputs=inputs, outputs=outputs)
  model.compile(
      optimizer=keras.optimizers.Adam(1e-2),
      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
      metrics=[keras.metrics.SparseCategoricalAccuracy()])

  model.summary(print_fn=logging.info)
  return model

# TFX Trainer will call this function.
def run_fn(fn_args: tfx.components.FnArgs):
  """Train the model based on given args.

  Args:
    fn_args: Holds args used to train the model as name/value pairs.
  """

  # This schema is usually either an output of SchemaGen or a manually-curated
  # version provided by pipeline author. A schema can also derived from TFT
  # graph if a Transform component is used. In the case when either is missing,
  # `schema_from_feature_spec` could be used to generate schema from very simple
  # feature_spec, but the schema returned would be very primitive.
  schema = schema_utils.schema_from_feature_spec(_FEATURE_SPEC)

  train_dataset = _input_fn(
      fn_args.train_files,
      fn_args.data_accessor,
      schema,
      batch_size=_TRAIN_BATCH_SIZE)
  eval_dataset = _input_fn(
      fn_args.eval_files,
      fn_args.data_accessor,
      schema,
```

```
    batch_size=_EVAL_BATCH_SIZE)

  model = _build_keras_model()
  model.fit(
    train_dataset,
    steps_per_epoch=fn_args.train_steps,
    validation_data=eval_dataset,
    validation_steps=fn_args.eval_steps)

  # The result of the training should be saved in `fn_args.serving_model_dir`
  # directory.
  model.save(fn_args.serving_model_dir, save_format='tf')

def _create_pipeline(pipeline_name: str, pipeline_root: str, data_root: str,
                     module_file: str, serving_model_dir: str,
                     metadata_path: str) -> tfx.dsl.Pipeline:
  """Creates a three component penguin pipeline with TFX."""
  # Brings data into the pipeline.
  example_gen = tfx.components.CsvExampleGen(input_base=data_root)

  # Uses user-provided Python function that trains a model.
  trainer = tfx.components.Trainer(
      module_file=module_file,
      examples=example_gen.outputs['examples'],
      train_args=tfx.proto.TrainArgs(num_steps=100),
      eval_args=tfx.proto.EvalArgs(num_steps=5))

  # Pushes the model to a filesystem destination.
  pusher = tfx.components.Pusher(
      model=trainer.outputs['model'],
      push_destination=tfx.proto.PushDestination(
          filesystem=tfx.proto.PushDestination.Filesystem(
              base_directory=serving_model_dir)))

  # Following three components will be included in the pipeline.
  components = [
      example_gen,
      trainer,
      pusher,
  ]

  return tfx.dsl.Pipeline(
      pipeline_name=pipeline_name,
      pipeline_root=pipeline_root,
      metadata_connection_config=tfx.orchestration.metadata
      .sqlite_metadata_connection_config(metadata_path),
      components=components)

tfx.orchestration.LocalDagRunner().run(
  _create_pipeline(
      pipeline_name=PIPELINE_NAME,
      pipeline_root=PIPELINE_ROOT,
      data_root=DATA_ROOT,
      module_file=_trainer_module_file,
      serving_model_dir=SERVING_MODEL_DIR,
      metadata_path=METADATA_PATH))
```

Quantisation

08 November 2021 08:44

```
import tensorflow as tf

saved_model_dir = "/path/to/mobilenet_v1_1.0_224/"
converter = tf.lite.TFLiteConverter.
    from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [
    tf.lite.constants.FLOAT16]
tflite_model = converter.convert()

open("converted_model.tflite", "wb").write(tflite_model)
```

Post training API

- **Reduced float**
  - Just set flags - No data needed.
  - All float32 parameters change to float16
  - Operations are executed in float16 depending on hardware support (e.g. GPU has support)

```
import tensorflow as tf

saved_model_dir = "/path/to/mobilenet_v1_1.0_224/"
converter = tf.lite.TFLiteConverter.
    from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

open("converted_model.tflite", "wb").write(tflite_model)
```

Post training API

- **Hybrid quantization**
  - Just set flags - No data needed.
  - Parameters change to a quantized-int8 and float32 representation
  - Execution varies depending on hardware support (e.g. CPU kernels support it)
  - Operations without a hybrid specification are left in the original precision (e.g. float32)

```
import tensorflow as tf

saved_model_dir = "/path/to/mobilenet_v1_1.0_224/"
converter = tf.lite.TFLiteConverter.
    from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
def data_generator():
  for i in range(calibration_steps):
    # get sample input data
    yield [input_sample]
converter.representative_dataset = data_generator
tflite_model = converter.convert()
open("converted_model.tflite", "wb").write(tflite_model)
```

Post training API

- **Integer quantization**
  - Small amount of data needed - e.g. 100 samples
  - Parameters change to a quantized int8 representation
  - Execution varies depending on hardware support (e.g. CPU, , DSP, NPU kernels support it)
  - Operations without an integer-only specification are left in the original precision (e.g. float32)

Functional APIs

25 November 2021 11:12

```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, Flatten

input = Input(shape = (28,28))
x = Flatten()(input)
x = Dense(128, activation = 'relu')
predictions = Dense(10, activation = 'softmax')

from tensorflow.keras.models import Model

func_model = Model(inputs = inputs, outputs = predictions)
```

BRANCHING MODELS

Inception type model

```
layer_2_1 = Dense(32)(layer1)
layer_2_2 = Dense(32)(layer1)
layer_2_3 = Dense(32)(layer1)
layer_2_4 = Dense(32)(layer1)
merge = Concatenate([layer2_1, layer2_2, layer2_3, layer2_4])
```

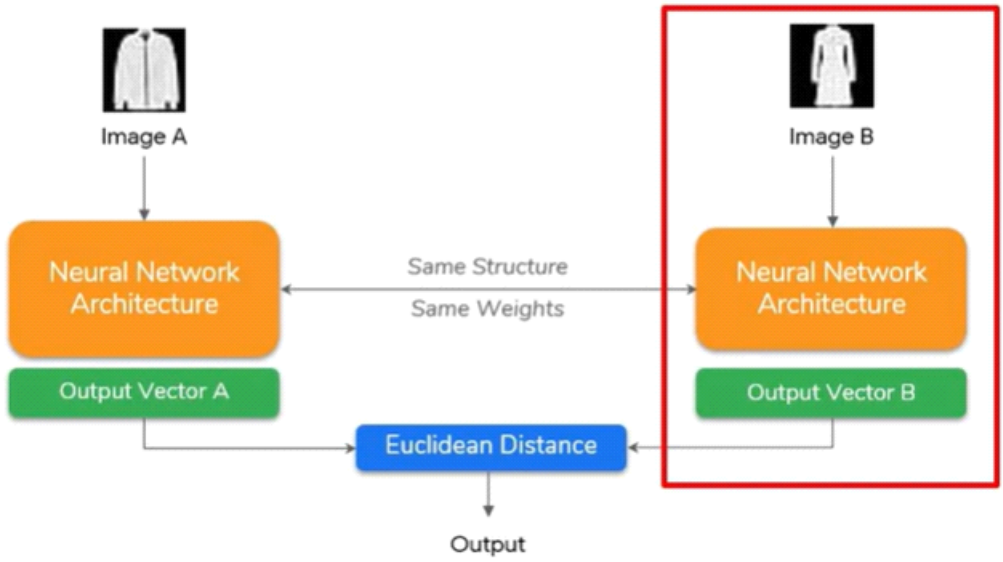
Multiple inputs and outputs

```
func_model = Model(inputs = [input1, input2], outputs = [output1, output2])
```

Creating a Multi-Output Model

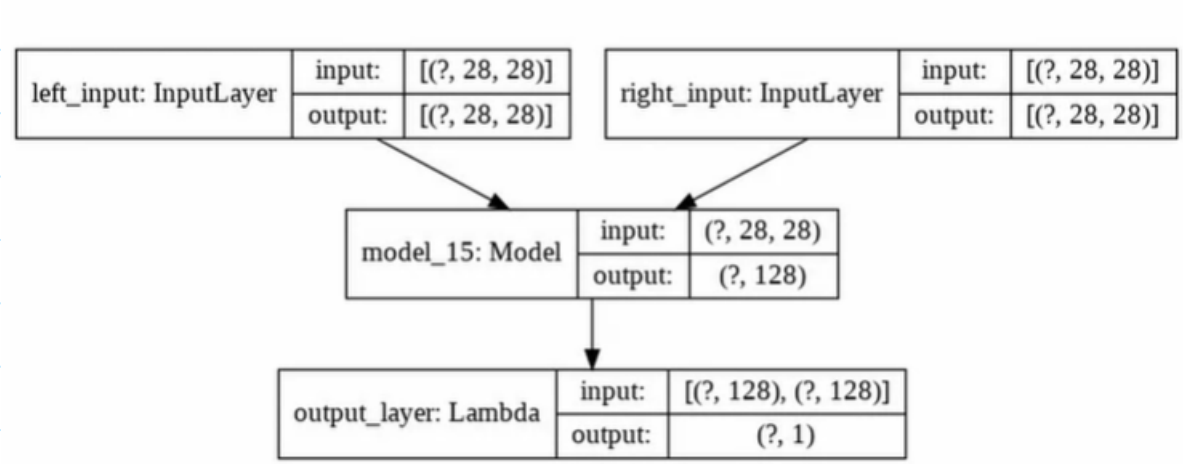
```
input_layer = Input(shape = (len(train.columns), ))
first_dense = Dense(128, activation = 'relu')(input_layer)
second_dense = Dense(128, activation = 'relu')(first_dense)

y1_output = Dense(1, name = 'y1_output')(second_dense)
third_dense = Dense(64, activation = 'relu')(second_dense)
y2_output = Dense(1, name = 'y2_output')(third_dense)

model = Model(inputs = input_layer, outputs = [y1_output, y2_output])

optimizer = tf.keras.optimizer.SGD(lr = 0.01)
model.compile(optimizer = optimizer, loss = {'y1_output':'mse', 'y2_output': 'mse'}, metrics = {'y1
_output': tf.keras.metrics.RootMeanSquaredError(), 'y2_output':
tf.keras.metrics.RootMeanSquarredError()})
from tensorflow.keras.layers import Layer
model._layers = [layer for layer in model._layers if isinstance(layer, Layer)]

plot_model(model)
plot_model(model, show_shapes = True, show_layer_names = True, to_file = 'model.png')

history = model.fit(norm_train_X, train_Y, epochs = 2000, batch_size = 10, validation_data =
(norm_test_X, test_Y))
```

#Siamese network: A Multiple Input Model




```
def create_pairs(x, digit_indices):
        """
        Positive and negative pair creation
        """
        pairs = []
        labels = []
        n = min([len(digit_indices[d]) for d in range(10)]) - 1
        for d in range(10):
                for i in range(n):
                        z1, z2 = digit_indices[d][i]. digit_indices[d][i+1]
                        pairs += [[x[z1], x[z2]]]
                        inc = random.randrange(1,10)
                        dn = (d + inc) % 10
                        z1, z2 = digit_indices[d][i], digit_indices[dn][i]
                        pairs +=[[x[z1], x[z2]]]
                        labels += [1,0]

        return np.array(pairs), np.array(labels)

def create_pairs_on_set(images, labels):
        digit_indices = [np.where(labels == i)[0] for i in range(10)]
        pairs, y = create_pairs(images, digit_indices)
        y = y.astype('float32')
        return pairs, y

def initialize_base_network():
        input = Input(shape = (28,28))
        x = Flatten()(input)
        x = Dense(128, activation = 'relu')(x)
        x = Dropout(0.1)(x)
        x = Dense(128, activation = 'relu')(x)
        x = Dropout(0.1)(x)
        x = Dense(128, activation = 'relu')(x)
        return Model(inputs = input, outputs = x)

def euclidean_distance(vects):
```

```
        x, y = vects
        sum_square = K.sum(K.square(x-y), axis = 1, keepdims = True)
        return K.sqrt(K.maximum(sum_square, K.epsilon()))

def eucl_dist_output_shape(shapes):
        shape1, shape2 = shapes
        return (shape1[0], 1)

base_network = initialize_base_network()

input_a = Input(shape = (28,28,), name = 'left_input')
vect_output_a = base_network(input_a)

input_b = Input(shape = (28,28,), name = 'right_input')
vect_output_b = base_network(input_b)

output = Lambda(euclidean_distance, output_shape = eucl_dist_output_shape)([vect_output_a,
vect_output_b])

model = Model([input_a, input_b], output)
```




```
def contrastive_loss_with_margin(margin):
        def contrastive_loss(y_true, y_pred):
                square_pred = K.square(y_pred)
                margin_square = K.square(K.maximum(margin - y_pred, 0))
                return K.mean(y_true * square_pred + (1 - y_true) * margin_square)
        return contrastive_loss

rms = RMSprop()
model.compile(loss = contrastive_loss, optimizer = rms)

model.fit([tr_pairs[:0], tr_pairs[:,1]], tr_y, epochs = 20, batch_size = 128, validation_data =
([ts_pairs[:,0], ts_pairs[:,1]], ts_y))

def compute_accuracy(y_true, y_pred):
        pred = y_pred.ravel() > 0.5
        return np.mean(pred == y_true)
```

Creating A Loss Function

25 November 2021 14:48

```
def my_loss_function(y_true, y_pred):
    # Huber loss
    threshold = 1
    error = y_true - y_pred
    is_small_error = tf.abs(error) <= threshold
    small_error_loss = tf.square(error) /2
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

$$L_\delta(a) \begin{cases} \frac{1}{2}a^2 & for|a| \leq \delta \\ \delta \times (|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

https://en.wikipedia.org/wiki/Huber_loss